
LeG
ANAYA
I. O. Angell - B. J. Jones
Diseño de gráficos
y videojuegos
Tratamiento en tres dimensiones
ZX SPECTRUM

DISEÑO DE GRAFICOS Y VIDEOJUEGOS

Tratamiento en tres dimensiones

Diseño de gráficos y videojuegos

Tratamiento en tres dimensiones

I. O. Angell
B. J. Jones



MICROINFORMATICA

Título de la obra original:
«ADVANCED GRAPHICS WITH THE SINCLAIR ZX SPECTRUM»

Traducción de: José Manuel Otón
Diseño de colección: Antonio Lax
Diseño de cubierta: Narcís Fernández

Publicado en Gran Bretaña por Macmillan Press
con el título «Advanced Graphics with the
Sinclair ZX Spectrum»

Villafranca, 22. 28028 Madrid
Depósito Legal: M. 10.123-1985
I.S.B.N.: 85-7614-006-1
Printed in Spain
Imprime: Gráficas FUTURA, Soc. Coop. Ltda.
Villafranca del Bierzo, 21-23. Pol. Ind. Cobo Calleja
Fuenlabrada (Madrid)

Indice

Prólogo

El rápido avance experimentado en los últimos años en la tecnología de ordenadores ha venido acompañado por una drástica reducción de su precio. Está asimismo prevista para un futuro próximo una reducción también notable en el precio de las unidades periféricas. Cabe pensar, por tanto, que un usuario de presupuesto limitado, que con anterioridad tenía acceso tan sólo a modelos muy elementales, podrá en lo sucesivo contar con ordenadores muy sofisticados. Podrá además librarse de las limitaciones que introduce el hecho de disponer como única salida del sistema una lista en forma de tabla numérica: por el contrario, estarán a su alcance unidades dotadas de microprocesador adaptables a un monitor de televisión, o modelos especiales dedicados a gráficos en color de bajo precio. En este sentido, el líder del sector ha sido siempre la casa Sinclair.

Sin embargo, los paquetes de programas y aplicaciones (todo lo que se agrupa bajo el nombre de *software* o logical) no parecen mostrar tendencia alguna de disminución de precio.

En el pasado, la obtención de salidas gráficas requería enormes inversiones tanto en *software* como en *hardware;* esta área, consecuentemente, ha sido reservada desde siempre a grandes grupos de investigación. Su inaccesibilidad al gran público ha hecho crecer alrededor del campo una cierta mística, atribuyéndosele injustamente una falsa reputación de dificultad. Este libro pretende alejar el fantasma de la complejidad; pretende también mostrar que los paquetes de *software,* complicados y caros, que por supuesto resultan de gran valor para un centro de investigación, no tienen por qué asustar al usuario normal: de hecho, son innecesarios para la mayor parte de las aplicaciones.

Este libro, por ejemplo, además de ser una introducción al tema de gráficos por ordenador, puede considerarse como un paquete de *software;* un paquete, además,

realmente barato: mucho más barato desde luego que cualquier aplicación que se haya comercializado. Evidentemente, debido a la naturaleza fundamentalmente básica de los contenidos del libro, el usuario deberá adquirir un conocimiento aceptable de su dispositivo gráfico antes de realizar dibujos distintos de los que se ofrecen como ejemplo. Esto no tiene por qué ser una desventaja: como se comprobará, el esfuerzo requerido es bastante limitado. Además, el conocimiento del usuario se irá acrecentando a medida que avanza en el libro, haciendo más difícil un error de interpretación en alguno de los subprogramas.

Se supone que el lector posee un conocimiento elemental de geometría cartesiana, así como del lenguaje de programación BASIC. Se presenta en el libro un gran número de ejercicios de programación que ayudarán al lector a aumentar su experiencia en BASIC. Por otra parte, el lenguaje BASIC es enormemente popular, de modo que todos los tipos de microordenadores disponen de él de una u otra forma. Los programas presentados aquí pueden por tanto ajustarse fácilmente para su ejecución en máquinas distintas del ZX Spectrum. Asimismo es un buen medio de transmisión de los algoritmos utilizados en gráficos de ordenador, permitiendo a los lectores traducir fácilmente estas ideas a otro lenguaje a su elección.

Los conceptos necesarios para el estudio de gráficos de ordenador están organizados como una combinación de teoría y ejemplos detallados. Estos últimos se introducen cuando y como se requieren para lograr una progresión normal de la materia. Se incluyen listados de programas como parte de esos ejemplos; no deben considerarse tan sólo algoritmos que describen la solución de problemas gráficos básicos, sino también como un paquete de *software* para gráficos, o simplemente como programas para dibujar determinados modelos. Estos ejemplos se completan con ejercicios que ayudan a ampliar las ideas expuestas.

A menudo, los problemas prácticos que van implícitos en la programación de gráficos son una fuente de dificultades mayor que los propios conceptos introducidos. Así pues, es fundamental que los lectores implementen el mayor número posible de los listados de programas que se dan, a fin de comprender los algoritmos, y que realicen también bastantes ejercicios. La cinta de cassette que acompaña al libro contiene la mayor parte de los listados mayores que se dan en el mismo. Si el lector se siente abrumado por las matemáticas es conveniente que comience por ejecutar los programas antes de comenzar a estudiar la teoría.

Este sistema ha sido utilizado con gran éxito en la enseñanza de gráficos por ordenador a estudiantes y posgraduados en el Royal Holloway College. La elaboración casi inmediata de dibujos aparentemente complicados genera un entusiástico interés en los alumnos. La habilidad adquirida de producción de dibujos de líneas y bocetos de color interactivos les deja una profunda huella, alcanzando muy pronto un alto nivel en el campo. No pretendemos llegar tan lejos con este libro, pero con su lectura podrá el lector interesado encontrar accesibles otros textos de mayor nivel.

Este libro está pensado para aquellos que son programadores competentes en BASIC pero novatos en el campo de gráficos. Contiene las ideas elementales y la información básica sobre *pixels* y gráficos bidimensionales, temas que deben ser dominados antes de emprender nuevas aventuras, como caracteres o gráficos tridimensionales. A continuación se presenta una sección dedicada a gráficos de caracteres y presentación de datos (en líneas y color), que es probablemente la aplicación co-

mercial no especializada más importante de los gráficos por ordenador. Los posteriores introducen al lector en la geometría del espacio tridimensional, y en una serie de proyecciones de dicho espacio en el espacio bidimensional que proporciona una pantalla gráfica. Se estudian con detalle los problemas relacionados con líneas y superficies ocultas, así como la construcción de objetos tridimensionales complicados. Finalmente, se ofrecen ideas avanzadas de programación BASIC y un ejemplo detallado de un videojuego.

Los gráficos constituyen una de las áreas de más rápida expansión en el mundo de los ordenadores. Se utiliza cada vez más en campos como Diseño Asistido por Ordenador (Computer Aided Design, CAD), Gestión Asistida por Ordenador y Enseñanza por Ordenador.

Hubo un tiempo en que tan sólo las grandes compañías, como los fabricantes de automóviles o aviones, utilizaban estas técnicas; en la actualidad, la mayoría de las empresas se han percatado del potencial y ahorro que suponen estas ideas. Pero no es sólo provechoso: ¡además es divertido! El Sinclair ZX Spectrum es una máquina ideal para aprender técnicas básicas de gráficos, y un excelente trampolín a unidades más complicadas (y más caras).

Esperamos que este libro muestre al menos en parte el entusiasmo que nosotros, nuestros colegas y nuestros estudiantes, sentimos por el tema. Para demostrar cuán útiles pueden ser estos dibujos para ilustrar libros o artículos, hemos realizado todas las figuras de este libro por ordenador.

Nota del traductor

En la versión inglesa de este libro, los nombres de los subprogramas tienen una doble finalidad, como nombre en sí y como etiqueta de llamada a subrutinas. Con el fin de no alterar la parte ejecutable de los programas originales, se han respetado dichas etiquetas. Al final de cada capítulo se incluyen las dos versiones, inglesa y española, de cada subrutina para su identificación. Al final del libro se presenta un índice de subrutinas en el que de nuevo se ofrece ambas versiones.

Introducción

Se puede leer este libro con una serie de criterios diferentes. En primer lugar, para aquellos que simplemente desean realizar dibujos complicados con su Spectrum, este libro puede ser simplemente un libro de recetas de programas gráficos; esperamos, naturalmente, que el lector, una vez realizadas estas figuras, se sienta inclinado a profundizar más en el libro, intentando comprender cómo y por qué se han realizado los programas. En segundo lugar, algunos de los programas pueden utilizarse para la construcción de diagramas de datos (círculos porcentuales, histogramas y gráficos), de evidente aplicación en negocios o laboratorios. Por último —el principal objetivo que movió a la realización de este libro— es un texto introductorio al campo de gráficos por ordenador, que suministra al lector desde las nociones más elementales de la materia hasta soluciones a problemas complicados, como gráficos de caracteres, construcción de objetos tridimensionales y algoritmos de líneas y superficies ocultas.

Los programas que aparecen más adelante en el libro son demasiado complicados como para presentarse en forma de listados simples. Además observaremos que hay una gran cantidad de repeticiones en la utilización de algoritmos elementales. Por consiguiente, hemos abordado los programas desde una perspectiva de arriba abajo o modular, de tal forma que la solución de cada problema gráfico se plantea como una serie de soluciones a subproblemas. Estos subproblemas, por su parte, pueden a su vez dividirse en un conjunto de problemas a resolver (módulos); estos módulos se programan en forma de subrutinas BASIC. A cada uno se le asigna un identificador (en letras minúsculas) y está dedicado a resolver una subtarea particular. La totalidad de estos módulos puede combinarse para resolver el problema gráfico requerido. En general, no utilizamos sentencias del tipo GO SUB 600 para pasar de una a otra subtarea de diferentes algoritmos; en su lugar, hemos preferido asignar el identificador de la subrutina al valor de la dirección de comienzo de la misma

(por ejemplo, LET escena3 = 6000), lo que nos permite escribir sentencias tales como GO SUB escena3. Hemos utilizado letras minúsculas para estos identificadores de subrutina (y también para grupos de subrutina dentro del texto); todas las demás variables de los programas están escritas en letras mayúsculas para evitar confusiones.

El BASIC de Spectrum no permite pasar parámetros a las subrutinas. Los valores de los parámetros de entrada tienen que ser asignados en sentencias fuera de la subrutina, y se deben conocer previamente los nombres de los parámetros de salida si se desea que la subrutina funciones correctamente. Esto puede ser un inconveniente cuando se utilizan subrutinas escritas por otra persona. Es fundamental que el usuario conozca los nombres de los parámetros de entrada y salida; por consiguiente, en nuestras subrutinas utilizamos sentencias REM para especificar los parámetros de entrada y salida de cada una de ellas. Hemos numerado todos nuestros programas de tal forma que todas las sentencias de programa se sitúan en líneas que terminan en cero, y las sentencias REM en líneas que terminan en cualquier otra cifra de 1 a 9, con excepción del propio nombre de la subrutina. Las sentencias REM que contienen los parámetros de entrada y salida van siempre a continuación de la sentencia REM con el nombre de la subrutina en líneas que acaban en 1 y en 2 respectivamente. Los listados de programas que van incluidos en la cinta de cassette utilizan además códigos para destacar o cambiar el color de las distintas sentencias REM (véase capítulo 13). En aquellos casos en que hemos considerado que la propia palabra REM molesta a la legibilidad de la línea, hemos empleado los códigos necesarios para hacerla invisible. Hemos minimizado el número de sentencias REM en la cinta para poder incluir el máximo número posible de listados de programa. Se aconseja que el usuario, por su parte, complete estos listados añadiendo comentarios a su gusto, y guardando en su propia cinta el nuevo programa por medio de un SAVE.

Para aquellos que deseen únicamente ejecutar los programas, presentamos al final de cada capítulo una lista de programas completos, junto con valores adecuados para los datos. De hecho, es una buena idea para todos los lectores, incluidos los "serios", guardar (SAVE) los subprogramas de la cinta antes de abordar cada capítulo. Podrán de esta manera trabajar con los programas por medio de órdenes LOAD, MERGE y RUN, según vayan apareciendo aquéllos en el texto. La cinta de cassette que acompaña al libro contiene todos los listados principales del mismo, así como datos (BYTE) para los diagramas y juegos de caracteres utilizados en los últimos programas (que de otra forma, hubieran tenido que ser construidos por los propios lectores, con un gasto de tiempo considerable). Los programas están preparados para un Spectrum 48K: si usted posee una máquina de 16K deberá realizar las modificaciones que se especifican en el apéndice A.

Como ejemplo de la tónica que seguirá el libro, presentamos a continuación el programa requerido para realizar la figura I.1, un dibujo de líneas de un cuerpo de revolución en el cual han sido suprimidas todas las líneas ocultas. Este programa funciona por igual en ambos tipos de máquina.

El programa requiere la fusión (MERGE) de los listados 2.1("comienzo"), 2.2 (dos funciones FN X y FN Y), 2.3 ("fijaorigen"), 2.4 ("trazapunto") y 3.3 ("recorte"). Esta combinación subprogramas se denomina "rut1", y está preparada para dibujar figuras de líneas en la pantalla de televisión.

A "rut1" se le deben añadir los listados 3.4 ("ángulo"), 8.1 ("multiplicación3" e "identificación3"), 8.2 ("traslación3"), 8.3 ("escala3"), 8.4 ("rotación3"), 9.1 ("observación3"), y 9.2 ("programa principal"). Estas subrutinas, combinadas, constituyen "rut3", utilizado para transformar y observar objetos en el espacio tridimensional.

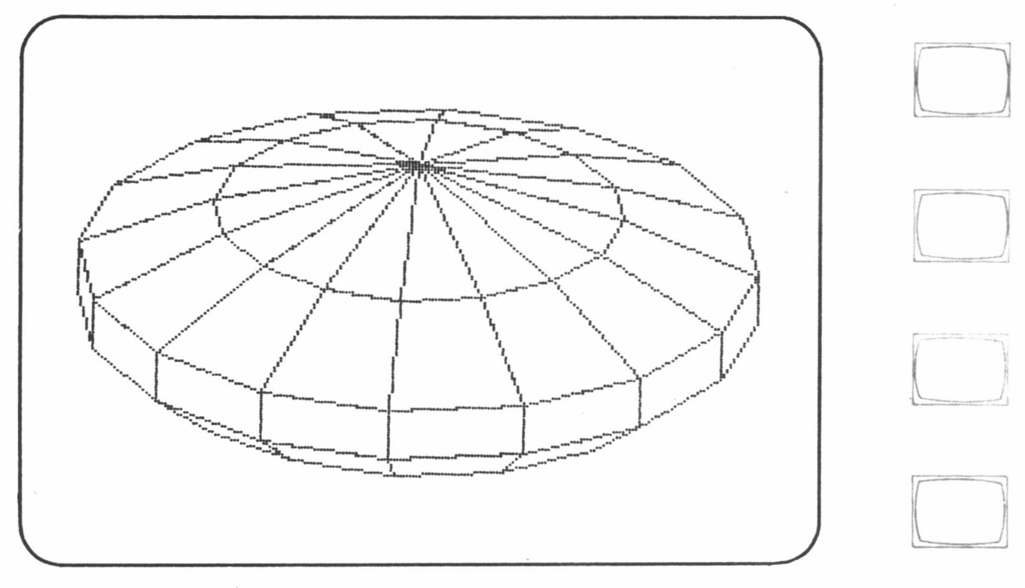


Figura I.1

También necesitamos añadir 10.3 ("cuerpo de revolución") así como la "escena3" que se muestra en el listado I.1 a continuación.

Listado I.1

```
6000 REM escena3/
 platillo volante
6010 DIM X(12): DIM Y(12)
6020 DIM S(6): DIM T(6)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6040 DATA 0,3, 3,2, 5,1, 5,0, 4,
-1, 0,-3
6050 RESTORE scene3
6060 LET revbod=6500
6069 REM Crea el objeto
6070 LET NUMV=5
6080 INPUT "NUMERO DE LINEAS HOR
IZONTALES",NUMH
```

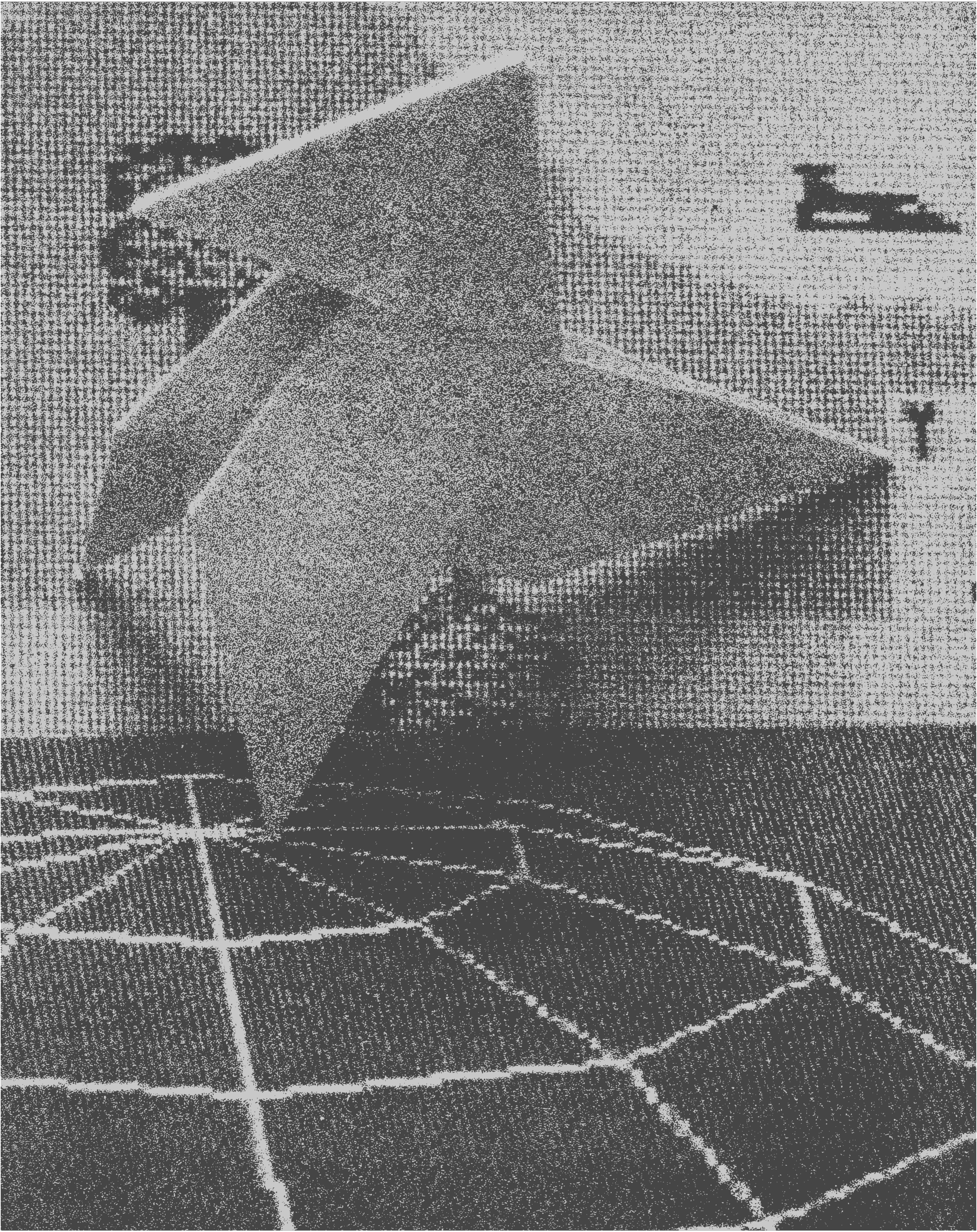
```
6090 INPUT "ANGULO FI ";PHI
6100 FOR I=1 TO NUMV+1: READ S(I
),T(I): NEXT I
6109 REM Posiciona al observador
6110 GO SUB idR3: GO SUB look3
6119 REM Dibuja el objeto
6120 GO SUB revbod
6130 RETURN
```

Para dibujar la figura I.1 se requieren los siguientes datos: HORIZ = 12, VERT = = 8, EX = 1, EY = 2, EZ = 3, DX = 0, DY = 0, DZ = 0, NUMH = 16 y PHI = 0. Cada valor debe ser tecleado independientemente, según los vaya solicitando la máquina. El dibujo tarda unos cinco minutos en realizarse, de manera que tenga paciencia. Ejecute el programa con diferentes datos de entrada. ¿Qué sucede si se hace HORIZ = = 6 y VERT = 4 manteniendo los demás valores constantes? Fíjese HORIZ = 15, VERT = 10, EX = 1, EY = −2, EZ = 3, DX = 1, DY = 0, y DZ = 0. Pruebe con NUMH = 20, PHI = 0.1. Necesitará leer hasta el capítulo 10 inclusive para comprender en su totalidad los detalles de lo que está sucediendo.

Este ejemplo ilustra el razonamiento que subyace detrás del propio libro. Suponiendo que usted sea un buen mecanógrafo, o que posea la cinta que acompaña al libro, podrá construir una figura tridimensional relativamente complicada de una forma muy rápida y sin "quemarse" demasiado. Incluso los "mecanógrafos" de un solo dedo (como los autores) encontrarán poca dificultad en implementar los programas presentados, aun antes de haber estudiado el libro con detalle.

Esperamos que este ejemplo le induzca a implementar todos los programas del libro, probar la mayoría de los ejemplos y a continuación intentar dibujar sus propios gráficos.

Lea ahora el resto del libro; le deseamos que pase muchas horas felices con su Spectrum.



1

Operaciones gráficas en el ZX Spectrum

A lo largo de este libro supondremos que el lector está razonablemente familiarizado con la programación en lenguaje BASIC en el ZX Spectrum. Sin embargo, en este capítulo, revisaremos algunos de los comandos BASIC; concretamente aquellos que están relacionados en todo o en parte con gráficos. Examinaremos las posibilidades del Spectrum con una serie de programas de ejemplo y ejercicios sencillos. En los capítulos siguientes utilizaremos este conocimiento para desarrollar una comprensión notable, tanto práctica como matemática, de los gráficos por ordenador.

Consideraremos en principio el *hardware* y *software* disponibles para la generación de dibujos. Todos los microordenadores capaces de producir una imagen en una pantalla de televisión generan sus representaciones gráficas utilizando tecnología RASTER SCAN. Esta circunstancia se da también en la mayor parte de los ordenadores mayores más modernos. En síntesis, reservamos un área de la memoria para guardar la información que se va a presentar en pantalla, y examinamos dicha área, bit a bit, a la vez que el pincel de electrones barre la pantalla. El resultado es una imagen compuesta de puntos, cada uno de ellos representado en la memoria por un único bit (un conmutador binario sí/no). En el caso más sencillo, el rayo se conecta durante un corto período de tiempo, cada vez que se encuentra un "sí" binario, produciendo un punto de luz en la pantalla.

INK y PAPER

El Spectrum nos ofrece dos comandos que controlan directamente la forma de presentación de los puntos. El dibujo está realizado con puntos de tinta *(ink)* correspondientes a los "síes" binarios (también llamados bits altos) sobre un fondo de

papel *(paper)* correspondiente a los "noes" binarios (bits bajos). Estos comandos u órdenes, llamados en el Spectrum PAPER e INK, se utilizan pulsando el nombre correspondiente seguido por un número N comprendido entre 0 y 9.

PAPER N inicializa el color de fondo del dibujo. Una vez ejecutada esta orden, todos los bits bajos que se generen en memoria se presentarán en pantalla en el color N (hasta que se ejecute una nueva orden PAPER).

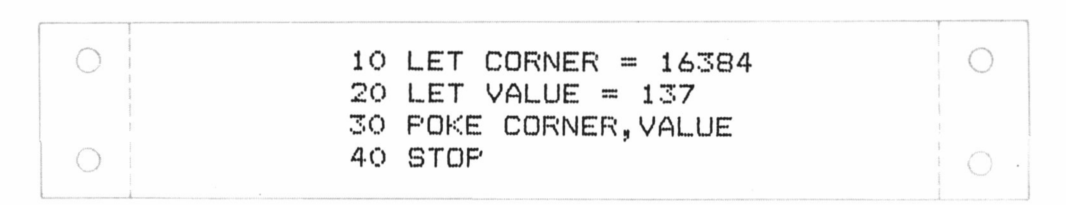
INK N inicializa los puntos luminosos correspondientes a los bits altos al color N de una forma semejante.

El número N, cuando está comprendido entre 0 y 7, representa el color impreso sobre la tecla correspondiente en el teclado del ordenador. Si se hace N igual a 9, el color de PAPER/INK se hace blanco o negro contrastando con el otro color INK/PAPER usado en ese momento. En general, tiene más claridad una tinta (INK) negra sobre papel (PAPER) blanco, como resulta obvio de la lectura de cualquier libro, siendo por tanto el utilizado normalmente.

Display File

Cuando existe una zona de la memoria que equivale a la pantalla, se dice que ésta está "cartografiada" *(mapped)*. En el Spectrum, esta zona de memoria se denomina *display file* y comienza en la localización 16384. Veamos cómo se afecta la salida en pantalla cuando cambiamos el contenido de esta zona de memoria. Podemos hacerlo con un programa como el presentado en el listado 1.1.

Listado 1.1



```
10 LET CORNER = 16384
20 LET VALUE = 137
30 POKE CORNER,VALUE
40 STOP
```

Este programa utiliza la industrucción POKE para almacenar un valor (VALUE) en la primera localización del *display file*. Esta localización contiene precisamente la información correspondiente a la esquina (CORNER) superior izquierda de la pantalla. La modificación conseguida es equivalente a la representación binaria del valor que hemos introducido como VALUE: en este caso, 10001001.

Ejercicio 1.1

1) Pruebe a introducir diferentes valores (VALUE), alterando el programa de la siguiente forma:

a) utilice una representación binaria (BIN) para VALUE.
b) utilice un bucle FOR ... NEXT para cambiar la variable.

2) Utilice los comandos PAPER e INK para cambiar los colores de fondo y escritura y vuelva a ejecutar el programa para observar las diferencias con el caso anterior.

BORDER

Al cambiar el color del papel (PAPER) observaremos inmediatamente que no podemos escribir en toda la pantalla del televisor. Por el contrario aparece un área alrededor del borde del papel, cuya misión es evitar la distorsión que existe en los extremos de la pantalla de cualquier televisor. El color de esta área externa se puede variar de una forma similar a la empleada con las órdenes PAPER e INK, por medio del comando

BORDER N

donde N puede variar entre 0 y 7 e indica el nuevo color del área externa.

Bloques de caracteres

Se puede conseguir un dibujo completo almacenando diferentes valores en las distintas localizaciones de memoria del *display file* de una forma semejante a la presentada en el listado 1.1. Por ejemplo, podemos almacenar los ocho valores 0, 98, 148, 136, 136, 136, 148, 98, en las localizaciones del *display file* que representan el comienzo de ocho líneas consecutivas de la pantalla (véase listado 1.2). En este caso, al igual que en el anterior, observaremos el dibujo formado por los puntos INK correspondientes a los "unos" mostrados en la figura 1.1.

	128	64	32	16	8	4	2	1	
00000000 =									= 0
01100010 =		64	+ 32			+ 4			= 98
10010100 =	128			+ 16		+ 4			= 148
10001000 =	128				+ 8				= 136
10001000 =	128				+ 8				= 136
10001000 =	128				+ 8				= 136
10010100 =	128			+ 16		+ 4			= 148
01100010 =		64	+ 32				+ 2		= 98

Figura 1.1

Adelantaremos que esta es la manera en que se definen (y redefinen) los caracteres en el Spectrum; comentaremos más detalladamente este tema en el capítulo 5. En cualquier caso, sirva este ejemplo para percatarnos de que un dibujo, aun tan

pequeño como éste, requiere bastante tiempo para su elaboración, debiendo realizarse un programa complicado en comparación con el resultado obtenido.

Listado 1.2

```
  10 LET CORNER = 16384
  20 LET LINE = 256
  30 DATA 0,98,148,136,136,136,1
36,148,98
  40 FOR I = 1  TO 7
  50 LET MEMORY = CORNER + I*LIN
E
  60 READ VALUE
  70 POKE MEMORY,VALUE
  80 NEXT I
  90 STOP
```

PLOT y DRAW

Hemos observado cómo se puede alterar la pantalla almacenando diferentes valores en el *display file.* Debemos tener en cuenta, sin embargo, que hay más de seis mil localizaciones en el mismo, y, por tanto, cambiar cada una de ellas individualmente puede resultar bastante tedioso. Necesitamos obviamente un método más efectivo de alteración de este *display.*

Dentro del BASIC se nos ofrece una serie de comandos gráficos para resolver este problema; los más sencillos son PLOT y DRAW. Todos los comandos gráficos funcionan considerando el *display* como una rejilla *(grid)* de 256 puntos horizontales por 176 verticales (45.056 en total). Estos puntos se denominan *pixels,* y se identifican por medio de una pareja de números enteros. Los comandos gráficos ayudan a construir dibujos permitiéndonos controlar una "pluma para gráficos" *(graphics pen),* que inicialmente está colocada en el *pixel* (0, 0). Veamos ahora cómo funcionan estos comandos.

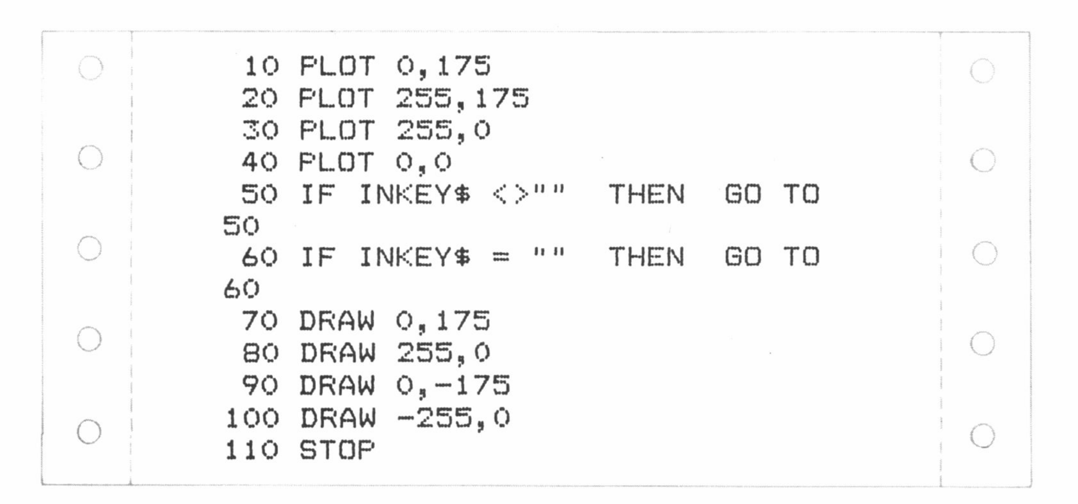
PLOT X, Y mueve nuestra pluma hasta el *pixel* (X, Y) y dibuja un punto allí.

DRAW X, Y dibuja una línea desde la posición actual de la pluma hasta el punto que se encuentra a X *pixels* de distancia en horizontal e Y *pixels* de distancia en vertical. Cuando X es negativo, el punto final se encontrará a la izquierda de la posición original, mientras que si es positivo, se hallará a la derecha. De igual forma, si Y es negativo, el punto final se localizará por debajo de nuestra posición anterior, y si es positivo, por encima.

Una vez ejecutada la orden, la pluma permanece en el último *pixel* alcanzado, esperando una nueva orden. Antes de pasar a estudiar los restantes comandos gráficos, más avanzados, observaremos lo que se puede hacer utilizando únicamente líneas rectas y/o puntos.

Estamos ahora en situación de dibujar figuras de gran tamaño en la pantalla. Por ejemplo, podemos dibujar un marco alrededor del área de pantalla accesible a gráficos (listado 1.3).

Listado 1.3



```
 10 PLOT 0,175
 20 PLOT 255,175
 30 PLOT 255,0
 40 PLOT 0,0
 50 IF INKEY$ <>""  THEN  GO TO
50
 60 IF INKEY$ = ""  THEN  GO TO
60
 70 DRAW 0,175
 80 DRAW 255,0
 90 DRAW 0,-175
100 DRAW -255,0
110 STOP
```

Este programa dibuja en primer lugar los puntos en la esquina del papel, utilizando el comando PLOT; a continuación espera a que se pulse una tecla y une por medio de líneas los puntos dibujados anteriormente. Al comparar el funcionamiento de los comandos PLOT y DRAW observamos que hay una importante diferencia entre ellos: el comando PLOT utiliza coordenadas de *pixel* absolutas, en tanto que DRAW usa las posiciones relativas de los puntos. Esto significa que, si se desea dibujar un segmento entre dos *pixels* de la pantalla, es necesario utilizar en primer lugar una orden PLOT para colocar la pluma en el punto de comienzo de la línea, calcular a continuación la posición del extremo opuesto con respecto al punto inicial, y finalmente utilizar la orden DRAW para dibujar la línea. Obsérvese que en el listado 1.3 están predeterminados todos los puntos antes de ejecutar el programa. Usualmente, los puntos se introducirán por medio de sentencias INPUT o READ, o bien se calcularán durante la ejecución.

Ejercicio 1.2

Escríbase un programa que calcule las posiciones de las líneas necesarias para dibujar un enrejado. Dibújese el mismo a continuación utilizando el comando DRAW dentro de dos bucles FOR ... NEXT (uno para las líneas horizontales y el otro para las verticales).

Ejercicio 1.3

Escríbase un programa que acepte N pares de coordenadas de *pixels* como INPUT desde teclado, y dibújese (usando DRAW) un polígono irregular de N lados uniendo los puntos introducidos por orden. Téngase en cuenta que el último punto introducido deberá unirse con el primero.

PRINT y LIST

Hasta ahora no hemos discutido el método más obvio de cambiar la pantalla, es decir, la utilización de los comandos PRINT y LIST. La razón es que estos comandos utilizan bloques del tamaño de un carácter, y están diseñados en principio para ser utilizados en gráficos de baja resolución. Volveremos en el capítulo 5 sobre este tema, pero, ya que el Spectrum permite mezclar libremente gráficos de alta y baja resolución, daremos un pequeño ejemplo aquí. Supóngase que añadimos la línea

```
5 LIST
```

al comienzo del programa del ejercicio 1.2, y que preparamos el programa para dibujar un enrejado de 32 líneas verticales y 22 horizontales. Obtendremos un resultado semejante al que se presenta en la figura 1.2, en el que se muestra el tamaño y la posición de los bloques de caracteres.

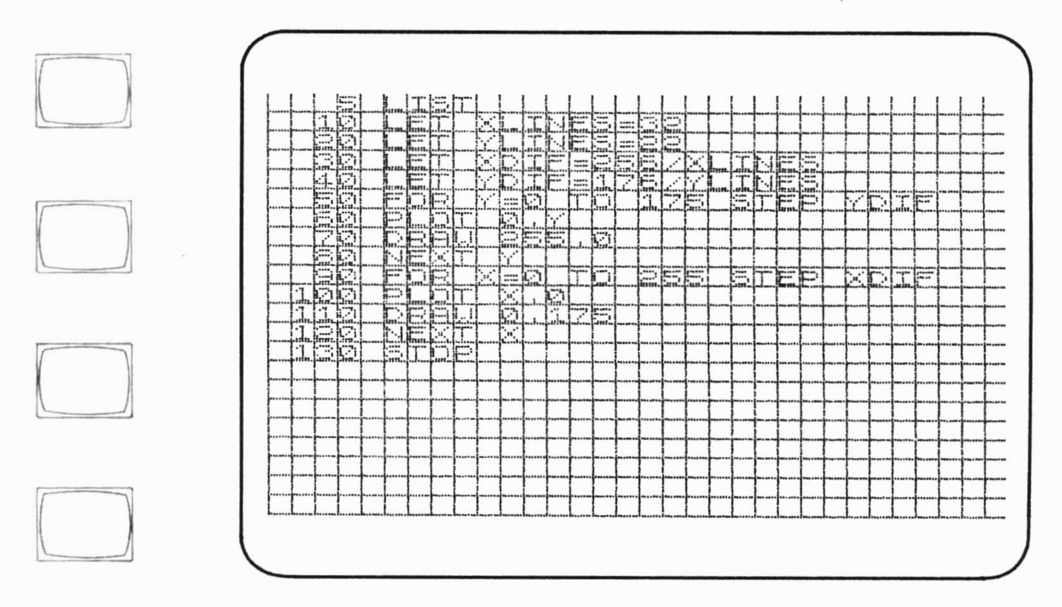


Figura 1.2

Podemos usar la orden PLOT para comprobar la capacidad de alta resolución del Spectrum dibujando curvas *fractales.*

Dibujar una fractal sencilla requiere seguir las siguientes etapas: imagínese un cuadrado cuyos lados tienen una longitud 4^n. Dicho cuadrado puede ser dividido en 16 cuadrados menores, cada uno de los cuales tiene una longitud 4^{n-1}. En la figura 1.3 hemos representado dichos cuadrados numerándolos desde el 1 al 16. Cambiando la posición de cuatro de estos cuadrados menores, los correspondientes a los números 2, 8, 9 y 15, obtenemos la figura 1.4.

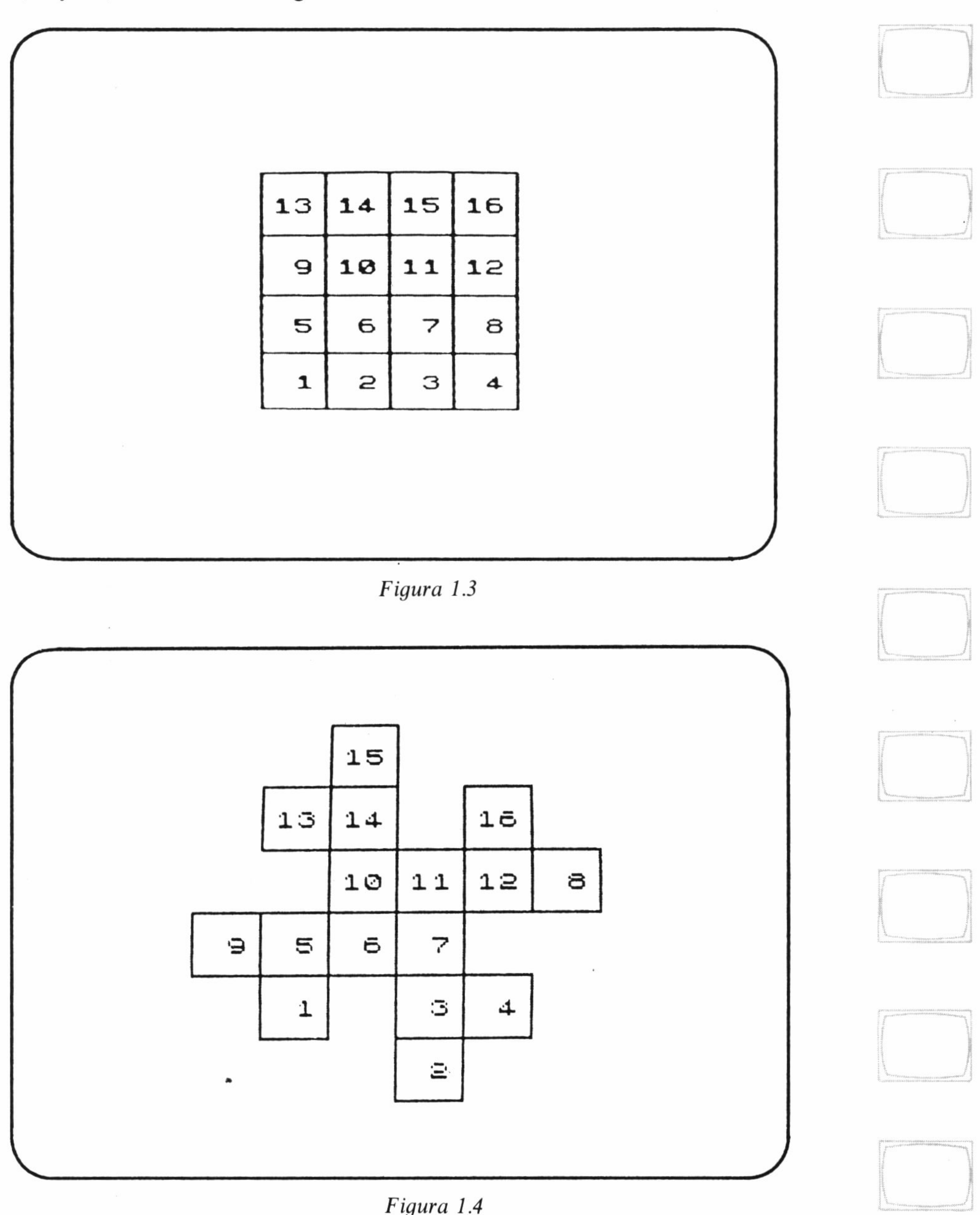


Figura 1.3

Figura 1.4

A continuación dividimos cada uno de los cuadrados del modelo obtenido en otros 16 cuadrados más pequeños y reordenamos los cuadraditos obtenidos en cada uno de ellos de una forma similar. Repetimos este proceso hasta que obtenemos cuadrados de longitud 1. La composición fractal resultante se compondrá exclusivamente de cuadrados unitarios, los cuales podemos dibujar (utilizando la orden PLOT) como *pixels* individuales. El programa que se presenta en el listado 1.4 comienza con un cuadrado cuyos lados tienen una longitud de 64 *pixels*, que equivale a 4^3; por tanto, debemos anidar 3 bucles FOR ... NEXT cada uno dentro del anterior. El resultado final se muestra en la figura 1.5.

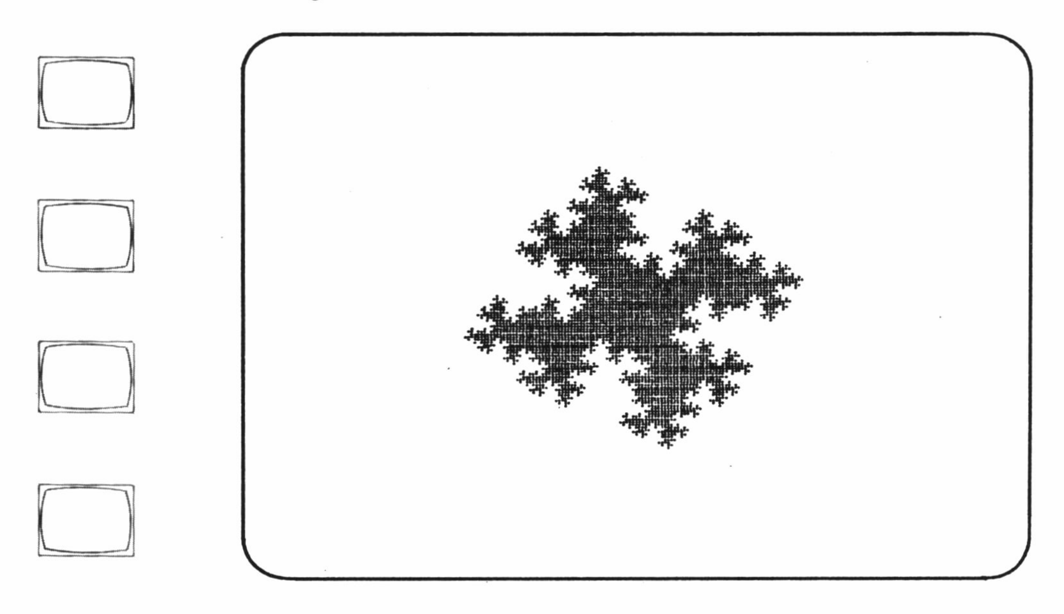


Figura 1.5

Listado 1.4

```
  10 DIM X(16):  DIM Y(16)
  20 FOR I = 1  TO 4
  30 FOR J = 1  TO 4
  40 LET K = 4*I + J - 4
  50 LET X(K) = J - 3:  LET Y(K)
= I - 3
  60 NEXT J:  NEXT I
  70 LET X(2)  =  0:  LET Y(2) =
  -3
  80 LET X(8)  =  2: LET Y(8) =
  0
```

```
 90 LET X(9)  = -3: LET Y(9) =
-1
100 LET X(15) = -1: LET Y(15)=
 2
110 FOR I = 1  TO 16
120 FOR J = 1  TO 16
130 FOR K = 1 TO 16
140 LET XX = 16*X(I) + 4*X(J) +
X(K)
150 LET YY = 16*Y(I) + 4*Y(J) +
Y(K)
160 PLOT 128+XX,88+YY
170 NEXT K:  NEXT J:  NEXT I
180 STOP
```

INVERSE y OVER

Estudiaremos ahora las distintas opciones que afectan al modo en que las líneas y puntos se presentan en la pantalla. Existen dos comandos de este tipo, los cuales se utilizan introduciendo el nombre del comando seguido por un número. El número es 1 para producir el efecto, y 0 para desconectarlo de nuevo.

INVERSE: cuando está conectado, todas las líneas o puntos que se dibujen aparecen en el color del fondo (PAPER). Es decir, los conmutadores binarios se desconectan en lugar de conectarse.

OVER: cuando se activa esta orden, cada *pixel* afectado por un comando gráfico cambia a su estado opuesto. Así un *pixel* de color de INK cambiará al color de PAPER y viceversa. Es decir, el conmutador binario cambia de posición cada vez que se afecta el *pixel.*

Se puede utilizar estas dos órdenes para producir programas que generan composiciones aparentemente complicadas y *displays* que cambian rápidamente. En el listado 1.5 se presenta un programa que combina dos métodos de creación de composiciones complicadas a partir de instrucciones muy simples.

Listado 1.5

```
  10 OVER 1
  20 LET LINES=400
  30 LET A=0: LET ANGLE=2*PI/LIN
ES
  40 FOR I=1 TO LINES
  50 LET X=85*COS A
  60 LET Y=85*SIN A
```

```
 70 PLOT 128,88
 80 DRAW X,Y
 90 LET A=A+ANGLE
100 NEXT I
110 OVER 0
120 STOP
```

Al estar formado el dibujo por una serie de puntos discretos *(pixels)*, las líneas inclinadas se dibujarán como una serie de pequeños escalones horizontales o verticales. Cuando se dibujan dos líneas inclinadas con ángulos ligeramente diferentes y suficientemente próximas la una a la otra, muchos de estos escalones coincidirán en ambas. Obsérvese la figura 1.6, dibujada por el listado 1.5. Las líneas que forman el área central solapan unas con otras muchas veces, de forma que dicha área debiera ser una mancha negra si no se utilizase la orden OVER. Utilizando OVER, sin embargo, aquellos *pixels* que han sido afectados por una orden de dibujo un número impar de veces quedarán conectados, en tanto que los otros se desconectarán. Esto produce el llamativo dibujo del centro de la figura. En la parte externa, por el contrario, el dibujo está formado por pequeños huecos residuales que aparecen entre los escalones de las líneas, tratándose por tanto de *pixels* que no pertenecen a ninguna de ellas.

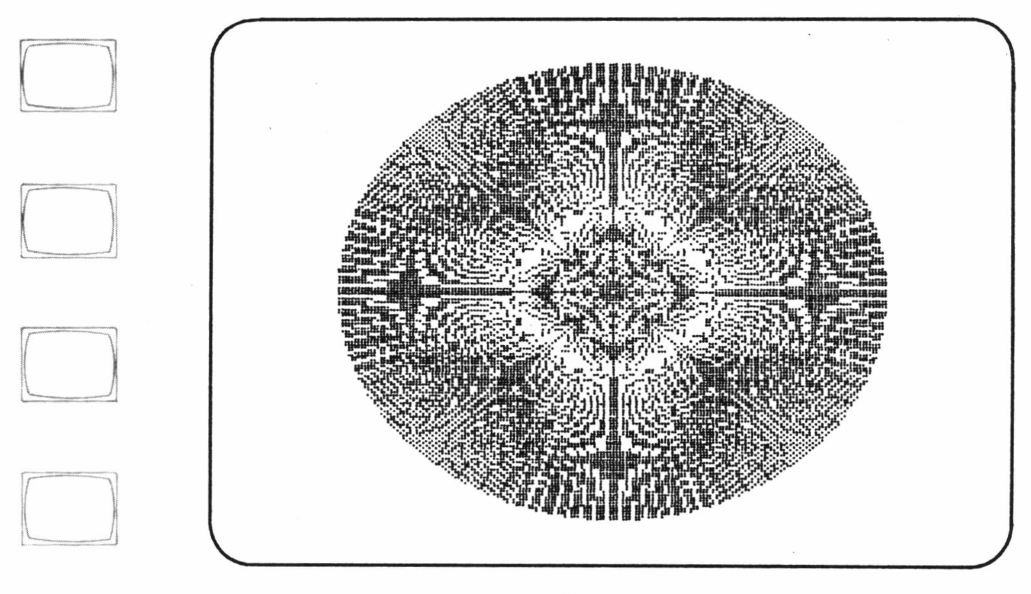


Figura 1.6

Ejercicio 1.4

Modifíquese el listado 1.5 para introducir el valor LINES por medio de una sentencia INPUT; introdúzcase también una variable de tipo *string,* que indique si se utiliza o no la opción OVER. Ejecute varias veces este programa variando el valor

LINES e incluyendo o no OVER para comprobar las partes del dibujo afectadas por una u otra variable.

Uno de los trucos más empleados para producir vistosas imágenes de formas cambiantes ha sido la utilización repetida de números aleatorios. El listado 1.6 presenta un sencillo ejemplo de este método utilizando RND y OVER para colocar *pixels* al azar en la pantalla.

Listado 1.6

```
10 OVER 1
20 LET X = INT (RND*256)
30 LET Y = INT (RND*176)
40 PLOT X,Y
50 BEEP 0.05,(X - Y)/10
60 GO TO 20
```

Ejercicio 1.5

Modifíquese el listado 1.6 de manera que dibuje líneas (DRAW). Realícese una versión que una los puntos aleatorios según se van generando, y otra que dibuje una línea desde el centro de la pantalla (128,88) a cada uno de los puntos.

En el ejercicio anterior observamos que la opción OVER asegura que el *display* cambiará cada vez, incluso si se repite el mismo comando; por ejemplo, dibujando el mismo punto o línea. La opción OVER se puede utilizar de esta forma para presentar en pantalla un objeto durante un corto período de tiempo, dibujándolo dos veces, la primera para presentarlo en la pantalla y la segunda para hacerlo desaparecer. El listado 1.7 mueve un punto alrededor de la pantalla dibujándolo con la orden PLOT e inmediatamente ejecutando otra orden PLOT en su posición anterior para eliminar el punto previo.

Listado 1.7

```
  10 OVER 1:  PAPER 0:  INK 7:
BORDER 4:  CLS
  20 LET SPEED = 2
  30 LET X = 0:  LET Y = 0
  40 LET XADD = SPEED:  LET YADD
 = SPEED
  50 PLOT X,Y
  60 LET OLDX = X:  LET OLDY = Y
```

```
  70 LET X = X + XADD
  80 IF X > 255 - SPEED  OR X <
SPEED  THEN  LET XADD = -XADD
  90 LET Y = Y + YADD
 100 IF Y > 174 - SPEED OR Y < S
PEED  THEN  LET YADD = -YADD
 110 PLOT X,Y
 120 PLOT OLDX,OLDY
 130 GO TO 60
```

Podemos ampliar este programa de manera que controlemos desde teclado el movimiento del punto listado 1.8). Las letras minúsculas alrededor de la “f” permiten al punto moverse en ocho direcciones distintas bajo nuestro control. Si se pulsa la letra “p” el punto va dejando un rastro que muestra sus movimientos anteriores. Si se pulsa una “q” el punto deja de marcar dicho rastro.

Este tipo de animación es una técnica comúnmente utilizada y de gran importancia. Por nuestra parte, haremos uso de ella con extensión, tanto en juegos como el incluido en el capítulo 14, por ejemplo, como en programas, como el “cursor” del capítulo 6.

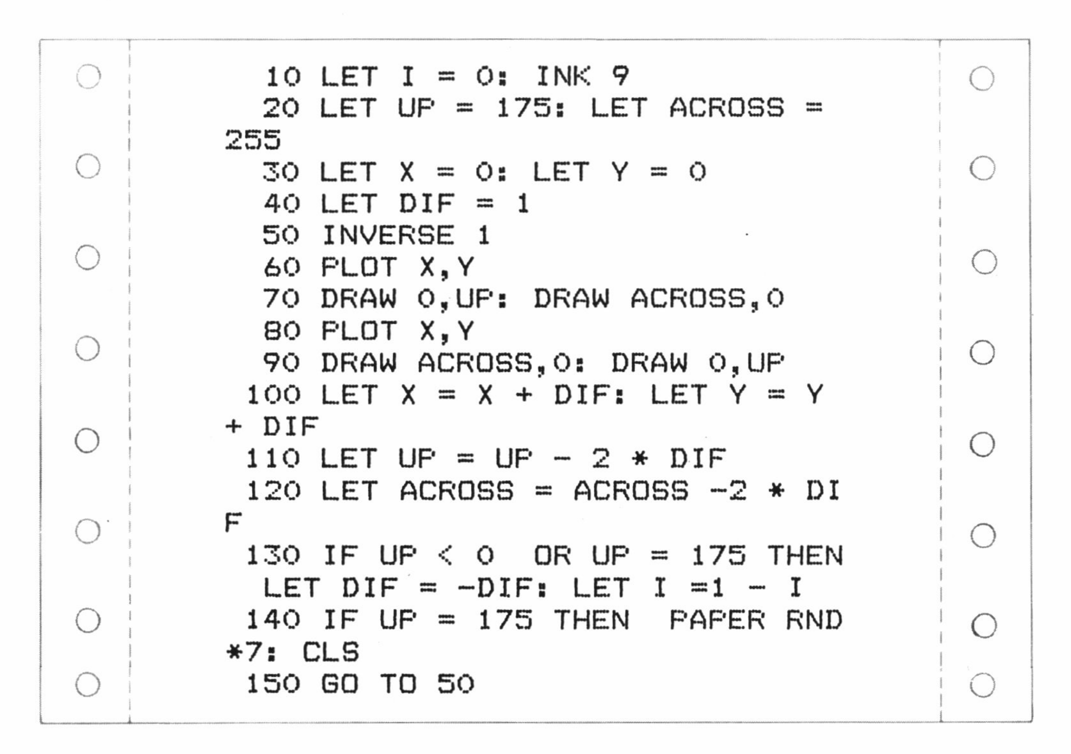
Listado 1.8

```
  10 OVER 1
  20 LET X = 0:  LET Y = 0
  30 PLOT X,Y
  40 LET OLDX = X:  LET OLDY = Y
  50 LET XADD = 0:  LET YADD = 0
  60 LET A$ = INKEY$:  IF A$ = "
" THEN  GO TO 60
  70 IF A$ = "p" THEN  OVER 0
  80 IF A$ = "q" THEN  OVER 1
  90 IF (A$ ="e"  OR A$ = "d" OR
 A$ = "c") AND X > 0 THEN  LET X
ADD = -1
 100 IF (A$ ="c"  OR A$ = "v" OR
 A$ = "b") AND Y > 0 THEN  LET Y
ADD = -1
 110 IF (A$ ="t"  OR A$ = "g" OR
 A$ = "b") AND X < 255 THEN  LET
 XADD = 1
 120 IF (A$ ="e"  OR A$ = "r" OR
 A$ = "t") AND Y < 175 THEN  LET
 YADD = 1
```

```
 130  IF XADD = 0 AND YADD = 0 T
HEN  GO TO 60
 140 LET X = X + XADD:  LET Y =
Y + YADD
 150 PLOT X,Y
 160 PLOT OLDX,OLDY
 170 GO TO 40
```

Podemos lograr secuencias de animación a gran escala utilizando líneas para extender o contraer un polígono. El listado 1.9 utiliza este método junto con la orden INVERSE para producir un efecto de "zoom" rápido.

Listado 1.9



```
  10 LET I = 0: INK 9
  20 LET UP = 175: LET ACROSS =
255
  30 LET X = 0: LET Y = 0
  40 LET DIF = 1
  50 INVERSE 1
  60 PLOT X,Y
  70 DRAW 0,UP: DRAW ACROSS,0
  80 PLOT X,Y
  90 DRAW ACROSS,0: DRAW 0,UP
 100 LET X = X + DIF: LET Y = Y
+ DIF
 110 LET UP = UP - 2 * DIF
 120 LET ACROSS = ACROSS -2 * DI
F
 130 IF UP < 0  OR UP = 175 THEN
  LET DIF = -DIF: LET I =1 - I
 140 IF UP = 175 THEN  PAPER RND
*7: CLS
 150 GO TO 50
```

Ejercicio 1.6

Dibuje y rellene un cuadrado compuesto por 40 × 40 *pixels*. Mueva dicho cuadrado en la pantalla controlando el movimiento por teclado. Nótese que necesita únicamente cambiar los bordes del cuadrado.

FLASH y BRIGHT

Habiendo visto ya lo que es posible hacer en blanco y negro, pasaremos ahora a comentar las posibilidades que tenemos para realizar gráficos en color. El Spectrum permite tener en pantalla todos los colores a la vez, pero dentro de cada bloque de caracteres pueden existir únicamente dos colores, los correspondientes a PAPER e INK. A estos colores se les puede asignar además el atributo BRIGHT y/o FLASH; además, se pueden conectar o desconectar de igual forma comandos como OVER e INVERSE para conseguir efectos especiales.

FLASH: cuando se asigna una orden FLASH a un bloque determinado, los colores PAPER e INK se intercambiarán alternativamente.

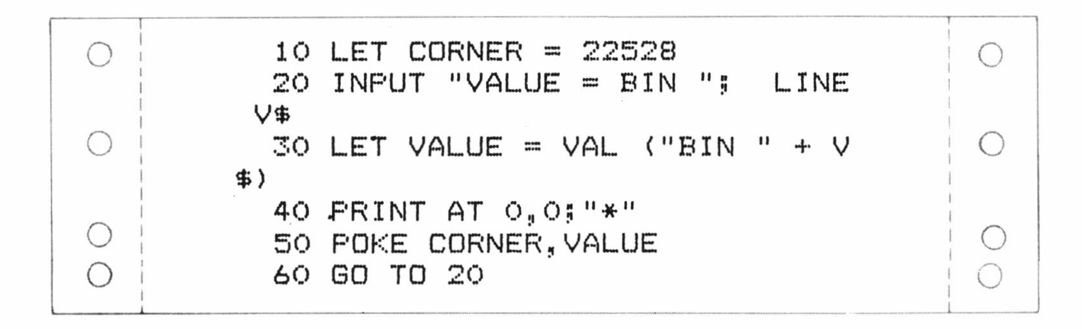
BRIGHT: los bloques en los que se asigne un atributo BRIGHT mostrarán un brillo incrementado tanto en el color del papel como de la tinta (PAPER e INK, respectivamente). Esto hace que el resto de bloques no afectados por dicho atributo aparezcan más oscuros.

FLASH y BRIGHT se pueden asignar también a 8, manteniéndose los valores preexistentes sin alteraciones de órdenes PRINT.

Atributos

Las combinaciones de FLASH, BRIGHT, PAPER e INK utilizadas para cada bloque se almacenan en memoria en un fichero denominado fichero de atributos *(attribute file)*. En dicho fichero se utiliza una localización de memoria para cada bloque; se encuentra situado en la memoria inmediatamente después del *display file,* comenzando en la localización 22528. El listado 1.10, una versión modificada del listado 1.1, altera estos valores directamente tal como hicimos con el *display file.*

Listado 1.10



```
10 LET CORNER = 22528
20 INPUT "VALUE = BIN "; LINE V$
30 LET VALUE = VAL ("BIN " + V$)
40 PRINT AT 0,0;"*"
50 POKE CORNER,VALUE
60 GO TO 20
```

Ejercicio 1.7

Utilice el programa del listado 1.10 para alterar bits individuales dentro de la variable VALUE almacenada en la primera localización del fichero de atributos.

Esta variable VALUE afecta al primer bloque completo de manera que los puntos pertenecientes a dicho bloque podrán parpadear o incrementar su intensidad, según estén activados FLASH y BRIGHT respectivamente; también estarán determinados los colores utilizados para PAPER e INK. Toda esta información se almacena en forma de un número binario (BIN) de la siguiente manera:

FLASH – 0 ó 1; BRIGHT – 0 ó 1; PAPER – 000 a 111; INK – 000 a 111

De esta forma podemos calcular el significado de introducir un determinado valor en el fichero de atributos, según se indica en la figura 1. .

		FLASH	BRIGHT	PAPER	INK
Valores normales	=	1	0	5	2
Equivalente BINario	=	1	0	101	010
BIN 10101010	=	128 + 32 + 8 + 2 = 170			

Figura 1.7

El ejemplo anterior asigna el valor adecuado a la localización correspondiente del fichero de atributos para que el bloque afectado parpadee (FLASH), no brille (no BRIGHT), en tanto que el fondo (PAPER) será de color cyan (azul claro), y la tinta (INK) roja. Se puede averiguar el valor de los atributos asignados a cualquier bloque utilizando la función ATTR (ROW, COLUMN). Los parámetros ROW y COLUMN especifican la posición del bloque contando el número de filas (ROW) a partir de la línea superior, y el número de columnas (COLUMN) a partir del borde izquierdo. De hecho, son los mismos parámetros que se utilizan en la orden PRINT AT para seleccionar un determinado bloque. De esta forma es muy sencillo escribir un programa que cambie los atributos de los bloques al azar (véase listado 1.11). El efecto conseguido es el mismo que el que se lograría introduciendo con órdenes POKE valores aleatorios en el fichero de atributos (véase Manual BASIC del Spectrum).

Listado 1.11

```
 10 FLASH  INT (RND*2)
 20 BRIGHT INT (RND*2)
 30 PAPER  INT (RND*8)
 40 INK    INT (RND*8)
 50 LET ROW = INT (RND*22): LET
COL = INT (RND*32)
 60 PRINT AT ROW,COL;"*"
 70 BEEP 0.02,RND*40
 80 GO TO 10
```

Ejercicio 1.8

Modifique el programa anterior de manera que cambie un bloque al azar, y a continuación calcule y presente en pantalla los atributos FLASH, BRIGHT, INK y PAPER impuestos, por medio de la función ATTR.

Por conveniencia, utilizamos la tabla 1.1 para efectuar la conversión entre valores asignados al fichero de atributos y su equivalente binario que será el que determine los atributos a utilizar.

Tabla 1.1. *Conversión de atributos*

PAPER	INK								MODO
	Black	Blue	Red	Magenta	Green	Yellow	Cyan	White	
Black	0	1	2	3	4	5	6	7	NORMAL
	64	65	66	67	68	69	70	71	BRIGHT
	128	129	130	131	132	133	134	135	FLASH
	192	193	194	195	196	197	198	199	BRIGHT + FLASH
Blue	8	9	10	11	12	13	14	15	NORMAL
	72	73	74	75	76	77	78	79	BRIGHT
	136	137	138	139	140	141	142	143	FLASH
	200	201	202	203	204	205	206	207	BRIGHT + FLASH
Red	16	17	18	19	20	21	22	23	NORMAL
	80	81	82	83	84	85	86	87	BRIGHT
	144	145	146	147	148	149	150	151	FLASH
	208	209	210	211	212	213	214	215	BRIGHT + FLASH
Magenta	24	25	26	27	28	29	30	31	NORMAL
	88	89	90	91	92	93	94	95	BRIGHT
	152	153	154	155	156	157	158	159	FLASH
	216	217	218	219	220	221	222	223	BRIGHT + FLASH
Green	32	33	34	35	36	37	38	39	NORMAL
	96	97	98	99	100	101	102	103	BRIGHT
	160	161	162	163	164	165	166	167	FLASH
	224	225	226	227	228	229	230	231	BRIGHT + FLASH
Cyan	40	41	42	43	44	45	46	47	NORMAL
	104	105	106	107	108	109	110	111	BRIGHT

Tabla 1.1. *Conversión de atributos*

PAPER	INK								MODO
	Black	Blue	Red	Magenta	Green	Yellow	Cyan	White	
	168	169	170	171	172	173	174	175	FLASH
	232	233	234	235	236	237	238	239	BRIGHT + FLASH
Yellow	48	49	50	51	52	53	54	55	NORMAL
	112	113	114	115	116	117	118	119	BRIGHT
	176	177	178	179	180	181	182	183	FLASH
	240	241	242	243	244	245	246	247	BRIGHT + FLASH
White	56	57	58	59	60	61	62	63	NORMAL
	120	121	122	123	124	125	126	127	BRIGHT
	184	185	186	187	188	189	190	191	FLASH
	248	249	250	251	252	253	254	255	BRIGHT + FLASH

Podemos imaginar un *pixel* en una pantalla de televisión a color como un conjunto de tres puntos luminosos muy próximos colocados en los vértices de un triángulo equilátero. Cada *pixel* dispone de un punto rojo, otro azul y otro verde, y las localizaciones del fichero de atributos se utilizan para iluminar estos tres diferentes colores. El *display file* indicará que un *pixel* determinado debe dibujarse con un color de tinta (INK) en concreto. Los tres bits más bajos (los bits 0, 1 y 2) del valor correspondiente al bloque a que pertenece ese *pixel* se utilizan para decidir cuáles puntos, verde, rojo y/o azul deben estar iluminados y cuáles no. Nuestros ojos contienen únicamente tres tipos de pigmentos sensibles al color (verde, rojo y azul, respectivamente). El cerebro, por su parte, toma las señales de estos puntos y las combina en un único punto de un color compuesto. De esta forma, si los tres últimos bits del atributo son 111, equivalentes al color 7, tenemos una mezcla de verde, rojo y azul. Esta mezcla corresponde a la luz blanca o, en nuestro caso, a tinta blanca. Podemos traducir los demás códigos de color obtenibles escribiéndolos en forma binaria según se muestra en la figura 1.8.

Los *pixels* del bloque que correspondan a la zona de fondo (PAPER) tendrán sus tres puntos iluminados o no de forma equivalente. Los bits que controlan el color del papel son el 3, 4 y 5. El bit 6 del atributo indica si el bloque está afectado o no por BRIGHT; cuando el atributo BRIGHT está activo, se aumenta la intensidad de todos los puntos iluminados. El bit 7, por su parte, cuando está activo, intercambia los colores de INK y PAPER. La velocidad de intercambio (es decir, la velocidad de intermitencia) depende de un valor interno del ordenador: en el Spectrum el cambio se produce aproximadamente cada tercio de segundo.

Color	Número	Eq. binario	Puntos iluminados
Negro *(Black)*	0	000	
Azul *(Blue)*	1	001	Azul
Rojo *(Red)*	2	010	Rojo
Magenta *(Magenta)*	3	011	Rojo + Azul
Verde *(Green)*	4	100	Verde
Cian *(Cyan)*	5	101	Verde + Azul
Amarillo *(Yellow)*	6	110	Verde + Rojo
Blanco *(White)*	7	111	Verde + Rojo + Azul

Figura 1.8

En general, podemos utilizar en gráficos de alta resolución dos colores, pero se debe proceder con cuidado. Téngase en cuenta que en un bloque determinado no puede haber más de dos colores simultáneamente. Excepto por esta limitación, se pueden conseguir gráficos de alta resolución completamente coloreados.

Ejercicio 1.9

Experimente con los diferentes colores utilizando los programas de este capítulo. Observará que hay ciertas combinaciones de colores que resultan demasiado complicadas para que un aparato de televisión normal pueda tratarlas. A menos que esté utilizando un monitor de gran calidad en lugar del televisor, obtendrá efectos muy curiosos de ondas moviéndose en la pantalla utilizando una combinación de colores llamativos en el programa del listado 1.5.

Animación simple

Podemos producir más efectos de animación en baja resolución utilizando colores y la orden FLASH. El listado 1.12 ejemplifica algunas técnicas interesantes de animación en color. La primera parte del programa es particularmente útil, ya que la pantalla no necesita mantenimiento una vez que se inicializa. El borde de la figura es una secuencia de bloques compuesta por bloques de papel rojo y tinta cian y bloques de papel cian y tinta roja, todos ellos con la orden FLASH activa. Al ejecutarse este programa observará un curioso efecto óptico que hace pensar que los bloques rojos y azules se están desplazando alrededor del borde.

Listado 1.12

```
10 FLASH 1: PAPER 2: INK 5
20 FOR I = 0  TO 1
30 FOR J =0  TO 15
40 PRINT AT 0,2*J + I; " "
50 PRINT AT 21,2*J + 1 - I; "
```

```
 60 NEXT J
 70 FOR J = 0  TO 10
 80 PRINT AT 2*J +I,0; " "
 90 PRINT AT 2*J + 1 -I,31; " "
100 NEXT J
110 PAPER 5: INK 2
120 NEXT I
130 FLASH 0: LET P = 0
140 FOR I = 1  TO 20
150 PRINT I,1; PAPER P;" "
160 LET P = P + 1: IF P = 7 THE
N  LET P = 0
170 NEXT I
180 GO TO 140
```

Ejercicio 1.10

Escriba una versión en color de baja resolución del programa del punto en movimiento o de cualquier otro programa de animación. Mueva en la pantalla un bloque completo en lugar de un *pixel.*

CIRCLE y DRAW

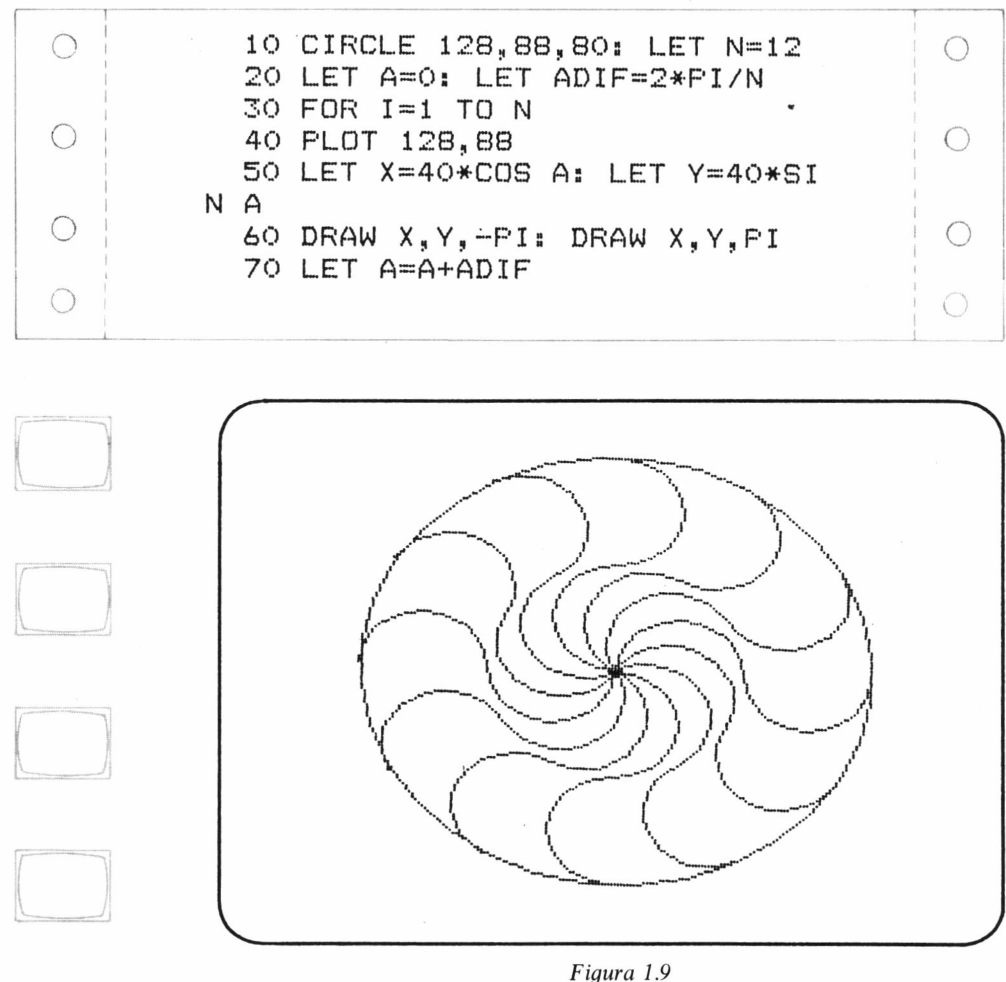
El Spectrum dispone de otros dos comandos para gráficas de alta resolución que no hemos estudiado todavía: CIRCLE y DRAW. Se utilizan ambos para dibujar líneas curvas.

Una orden del tipo CIRCLE X, Y, R dibuja un círculo de radio R *pixels* con el centro situado en el *pixel* (X, Y). Es importante tener en cuenta que una vez ejecutada esta orden, la pluma queda situada en el *pixel* situado en la parte derecha del círculo que está justo debajo del centro.

Por su parte, DRAW X, Y, A dibuja una línea curva desde la posición en que se encuentra la pluma en ese momento hasta la posición relativa X, Y, girando un ángulo A. Esta línea curva será un arco de círculo. El ángulo efectuado por la línea debe especificarse en radianes y estar comprendido entre —PI y PI.

Un ejemplo del uso de estos comandos se presenta en el listado 1.13, que producen la imagen de la figura 1.9.

Listado 1.13



```
10 CIRCLE 128,88,80: LET N=12
20 LET A=0: LET ADIF=2*PI/N
30 FOR I=1 TO N
40 PLOT 128,88
50 LET X=40*COS A: LET Y=40*SI
N A
60 DRAW X,Y,-PI: DRAW X,Y,PI
70 LET A=A+ADIF
```

Figura 1.9

Ejercicio 1.11

La figura 1.10 muestra un dibujo conocido como espiral celta o *triskele*. Escriba un programa que dibuje este tipo de modelo.

Colores dentro de un mismo bloque

La utilización de tintas (INK) de diferentes colores en gráficos de alta resolución puede producir problemas. Los resultados que se obtienen, aunque calculables, suelen ser inesperados. Ejecute el programa del listado 1.14. En él se puede comprobar la

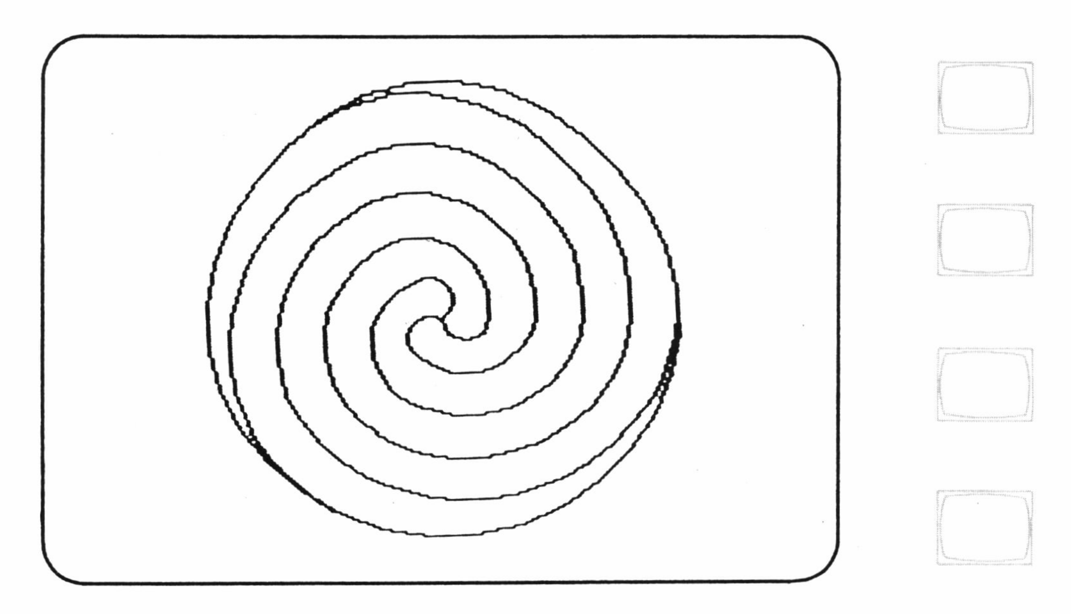
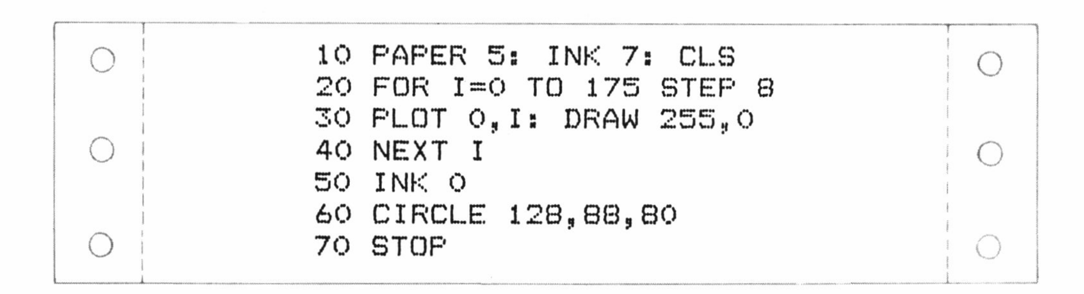


Figura 1.10

facilidad con que las cosas pueden ir mal cuando se utilizan más de dos colores sin un cuidadoso planteamiento previo.

Listado 1.14



```
10 PAPER 5: INK 7: CLS
20 FOR I=0 TO 175 STEP 8
30 PLOT 0,I: DRAW 255,0
40 NEXT I
50 INK 0
60 CIRCLE 128,88,80
70 STOP
```

Sin embargo, podemos aprovechar esta dificultad en nuestro beneficio. Podemos producir imágenes de color de baja resolución aparentemente complicadas y en rápido cambio. Inicialmente, imprimimos varios bloques en determinadas posiciones de la pantalla por medio de órdenes PRINT. Si a continuación dibujamos líneas que atraviesen estos bloques, se producirá un cambio en el color de los mismos. Esta técnica es de una velocidad realmente notable, según se puede observar ejecutando el programa del listado 1.15.

Listado 1.15

```
  10 INK 7: CLS
  20 FOR I=1 TO 704
  30 LET DIST=80: LET I=0: LET D
=8
  40 INK I
  50 PLOT 120,86+DIST: DRAW DIST
,-DIST: DRAW -DIST,-DIST
  60 PLOT 127,86-DIST: DRAW -DIS
T,DIST: DRAW DIST,DIST
  70 LET DIST=DIST-D: IF DIST=0
OR DIST=80 THEN  LET D=-D
  80 LET I=I+1: IF I=8 THEN  LET
 I=0
  90 GO TO 40
```

Un juego sencillo

A continuación presentamos un pequeño programa de juegos (listado 1.16) como ejemplo final de la utilización de las técnicas descritas en este capítulo. Disponemos de un gusano que se puede mover en etapas del tamaño de un bloque cada vez en toda la pantalla, horizontal o verticalmente, controlando su movimiento desde el teclado. El propósito del juego es conseguir que el gusano se coma el dinero simbolizado por una £. A medida que el gusano come, va creciendo. Si en algún momento la cabeza del gusano se estrella contra los bordes o contra su propio cuerpo, el gusano muere. Cada vez que se alcanzan diez fichas comidas, el gusano vuelve a su tamaño original, acompañado por una fanfarria. El juego continúa de igual manera.

El juego está desarrollado utilizando los métodos modulares estructurados que suelen preferir los programadores. Estos métodos ayudan a producir con rapidez un programa comprensible y funcionando. En síntesis, el programa está dividido en una serie de pequeñas tareas que en conjunto componen el juego completo. Para este juego las tareas a realizar son las siguientes:

A. Inicializar las variables.
B. Preparar el tablero.
C. Control del juego.
D. Actualización y tanteo.

Ahora podemos intentar resolver cada uno de los problemas planteados o, en caso necesario, subdividirlos a su vez en otros más pequeños que resulten fáciles de abordar.

Por ejemplo, la tarea C se puede subdividir en:

1. Generación de la moneda a comer.
2. Utilización del teclado para cambiar la dirección del gusano.
3. Movimiento del gusano.

A su vez, la subtarea C.3 se puede expresar como:

a) Dibujar el gusano.
b) Si el gusano alcanza el borde o a sí mismo, muere.
c) Fanfarria.

Este cuadro no implica un orden predeterminado: por ejemplo, puede darse el caso de que convenga realizar la regeneración de la moneda dentro de la parte de programa dedicada a la fanfarria. En realidad, estos apartados indican únicamente una lista de tareas a realizar que se nos hacen evidentes cuando intentamos resolver un problema de mayor cuantía.

Observe el listado del juego; intente identificar qué tareas se realizan, dónde, en qué orden y cuáles de ellas han sido subdivididas. (A lo largo del libro utilizaremos variables en letras minúsculas para expresar líneas de programa en que comienzan subrutinas. Así se hace el programa más legible, dando una clara idea del algoritmo; es una costumbre que aconsejamos.)

Nótese la utilización de expresiones lógicas (del tipo IF DEAD THEN..., por ejemplo); consúltese en caso necesario el Manual del Spectrum. Obsérvese también que se emplea ATTR y SCREEN$ para detectar colisiones, tanto por el color del carácter como por su contenido. En la figura 1.11 se puede observar el aspecto del juego en funcionamiento.

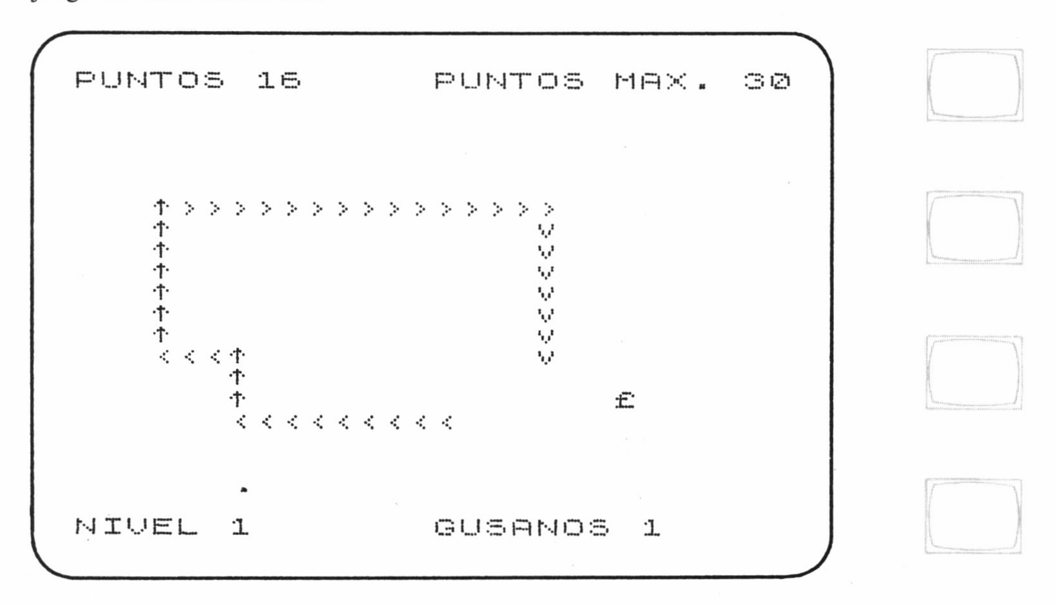


Figura 1.11

Listado 1.16

```
1000 DIM R(55): DIM C(55)
1009 REM Inicializa los punteros
 de rutina y pone la puntuacion
a cero
1010 LET fanfare=2000: LET worm=
3000: LET key=4000: LET gobble=5
000: LET status=6000: LET target
=7000
1020 BORDER 1: PAPER 7: INK 0
1030 LET HSC=0
1039 REM Comienzo del juego
1040 LET SCORE=0: LET WORMS=3: L
ET LEVEL=1
1048 REM Genera un gusano de 5 s
egmentos de longitud en fila R c
olumna C
1049 REM P es el puntero del seg
mento a mover
1050 LET S=5: LET P=1: LET R=0:
LET C=INT (RND*32)
1059 REM Fija las variables de m
ovimiento para que el gusano vay
a para abajo
1060 LET RMOVE=1: LET CMOVE=0: L
ET H$="v"
1069 REM Borra la matriz de posi
ciones del segmento
1070 FOR I=1 TO 55: LET R(I)=-1:
 NEXT I
1080 LET WON=0: LET DEAD=0
1089 REM Pinta la pantalla con f
ranjas amarillas
1090 CLS : PRINT AT 0,0; PAPER 6
;,,: PRINT AT 21,0; PAPER 6;,,
1099 REM Imprime la puntuacion
1100 GO SUB status: GO SUB targe
t
1109 REM Bucle principal del jue
go; comprueba los controles, mue
ve el gusano
1110 GO SUB key: GO SUB worm
1119 REM Repite el bucle
```

```
1120 IF NOT DEAD AND NOT WON  TH
EN  GO TO 1110
1129 REM Si el gusano se come 10
' genera otro gusano
1130 IF WON THEN  LET LEVEL=LEVE
L+1: GO SUB fanfare: GO TO 1050
1139 REM Hace ruido de choque
1140 IF DEAD THEN  FOR I=1 TO 10
: BEEP 0.005,15: NEXT I
1149 REM Genera otro gusano si q
uedan todavia
1150 IF DEAD THEN  LET WORMS=WOR
MS-1: IF WORMS<>0 THEN  GO TO 10
50
1160 FOR I=1 TO S: BEEP 0.01,0:
PRINT AT R(P),C(P); PAPER 7;" "
1170 LET P=P+1: IF P>S THEN  LET
 P=1
1180 NEXT I
1189 REM Renueva la puntuacion
1190 PRINT AT Y,X; PAPER 7;" ":
GO SUB status
1199 REM Espera hasta que se apr
ieta ENTER
1200 LET I$=CHR$ 18+CHR$ 1+"ENTE
R PARA EMPEZAR"+CHR$ 18+CHR$ 0
1210 INPUT (I$+"        "); LINE A
$
1220 GO TO 1040

2000 REM soniquete
2009 REM Musica tocada al pasar
un nivel
2010 DATA .06,18,.06,19,0.06,21,
0.15,27,0.06,21,0.2,27
2019 REM Lee y toca las combinac
iones de 6 notas y duraciones
2020 RESTORE fanfare: FOR I=1 TO
 6: READ L,T: BEEP L,T: NEXT I
2030 RETURN

3000 REM gusano
3008 REM El gusano se mueve colo
cando el ultimo segmento al prin
cipio
```

```
3010 IF R(P)<>-1 THEN  PRINT AT
R(P),C(P); PAPER 7;" "
3019 REM Calcula la posicion de
la cabeza del gusano
3020 LET R=R+RMOVE: LET C=C+CMOV
E
3029 REM Comprueba si hay colisi
on
3030 IF R>20 OR R<1 OR C>31 OR C
<0 THEN  LET DEAD=1: RETURN
3039 REM Comprueba si hay colisi
on con otro segmento del gusano
3040 IF ATTR (R,C)=16 THEN  LET
DEAD=1: RETURN
3049 REM Fija una nueva posicion
 de fila y columna del segmento
3050 LET R(P)=R: LET C(P)=C
3059 REM Comprueba si se ha comi
do el objeto
3060 IF SCREEN$ (R,C)="£" THEN
GO SUB gobble
3069 REM Pone el nuevo segmento
en pantalla
3070 PRINT AT R(P),C(P); PAPER 2
;H$
3080 LET P=P+1: IF P>S  THEN  LE
T P=1
3090 RETURN

4000 REM tecla

4009 REM Comprueba las teclas ap
retadas
4010 LET A$=INKEY$: IF A$="" THE
N  RETURN
4019 REM Comprueba que las letra
s son tratadas como minusculas
4020 IF CODE A$<96 THEN  LET A$=
CHR$ (CODE A$+32)
4028 REM El gusano no puede dars
e la vuelta sobre si mismo
4030 IF A$="i" AND CMOVE THEN  L
ET RMOVE=-1: LET CMOVE=0: LET H$
="^": RETURN
4039 REM abajo
```

```
4040 IF A$="m" AND CMOVE THEN  L
ET RMOVE=1: LET CMOVE=0: LET H$=
"v": RETURN
4049 REM izquierda
4050 IF A$="j" AND RMOVE THEN  L
ET RMOVE=0: LET CMOVE=-1: LET H$
="<": RETURN
4059 REM derecha
4060 IF A$="k" AND RMOVE THEN  L
ET RMOVE=0: LET CMOVE=1: LET H$=
">": RETURN
4070 RETURN

5000 REM engullido
5009 REM Come el objeto haciendo
 ruidos
5010 FOR I=2 TO 4 STEP 0.5
5020 BEEP 0.01,EXP I-10: NEXT I
5029 REM Incrementa la puntuacio
n
5030 LET SCORE=SCORE+1: GO SUB s
tatus
5039 REM Si el gusano tiene 55 s
egmentos ganas una vuelta
5040 LET S=S+5: IF S=55 THEN  LE
T WON=1: RETURN
5049 REM Situa un nuevo objeto e
n pantalla
5050 GO SUB target
5060 RETURN

6000 REM estado

6009 REM Si tu puntuacion es max
ima renueva esta
6010 IF SCORE>HSC  THEN  LET HSC
=SCORE
6019 REM Imprime las puntuacione
s
6020 PRINT AT 0,0; PAPER 6;"   P
UNTOS ";SCORE," PUNTOS MAX. ";HS
C
6030 PRINT AT 21,0; PAPER 6;"
NIVEL ";LEVEL," GUSANOS ";WORMS
6040 RETURN
```

```
7000 REM objeto

7009 REM Escoge un bloque de car
acter aleatoriamente
7010 LET X=RND*31: LET Y=RND*19+
1
7019 REM Comprueba que no es el
mismo lugar que el del anterior
7020 IF X=C AND Y=R THEN  GO TO
target
7029 REM Comprueba que no esta b
ajo el gusano
7030 IF ATTR (Y,X)=16 THEN  GO T
O target
7040 PRINT AT Y,X; PAPER 4;"£"
7050 RETURN
```

Ejercicio 1.12

Como proyecto final de este capítulo (miniproyecto más bien), intente escribir un juego de *squash* o de ping-pong (¡o ambos!) utilizando gráficos de baja resolución. Utilice como pelota un *pixel* o bloque completo, a su gusto, y controle la o las raquetas desde el teclado de la misma manera que se controla el gusano en el juego. Transforme parte del programa anterior en subrutinas de su propio programa a su conveniencia.

En este capítulo nos hemos limitado a utilizar la pantalla como una hoja de papel fija sobre la que hemos dibujado composiciones gráficas o juegos. Si deseamos pasar a dibujar objetos reales, disponiendo de *pixels* como única herramienta, deberemos buscar órdenes adecuadas que permitan reflejar el mundo real en nuestro mundo de *pixels*. Estudiaremos y desarrollaremos las técnicas necesarias para realizar estos dibujos reales en los siguientes capítulos.

Programas completos

I. Listado 1.1: no requiere datos.
II. Listado 1.2: no requiere datos.
III. Listado 1.3: no requiere datos.
IV. Listado 1.4: no requiere datos.
V. Listado 1.5: no requiere datos.
VI. Listado 1.6: no requiere datos.

VII. Listado 1.7: no requiere datos.

VIII. Listado 1.8: pulse las teclas que rodean a la "F" para mover el punto en cualquier dirección entre las 8 posibles. Pulse "P" para dejar un rastro o pista y "Q" para eliminarlo.

IX. Listado 1.9: no requiere datos.

X. Listado 1.10: no requiere datos.

XI. Listado 1.11: no requiere datos.

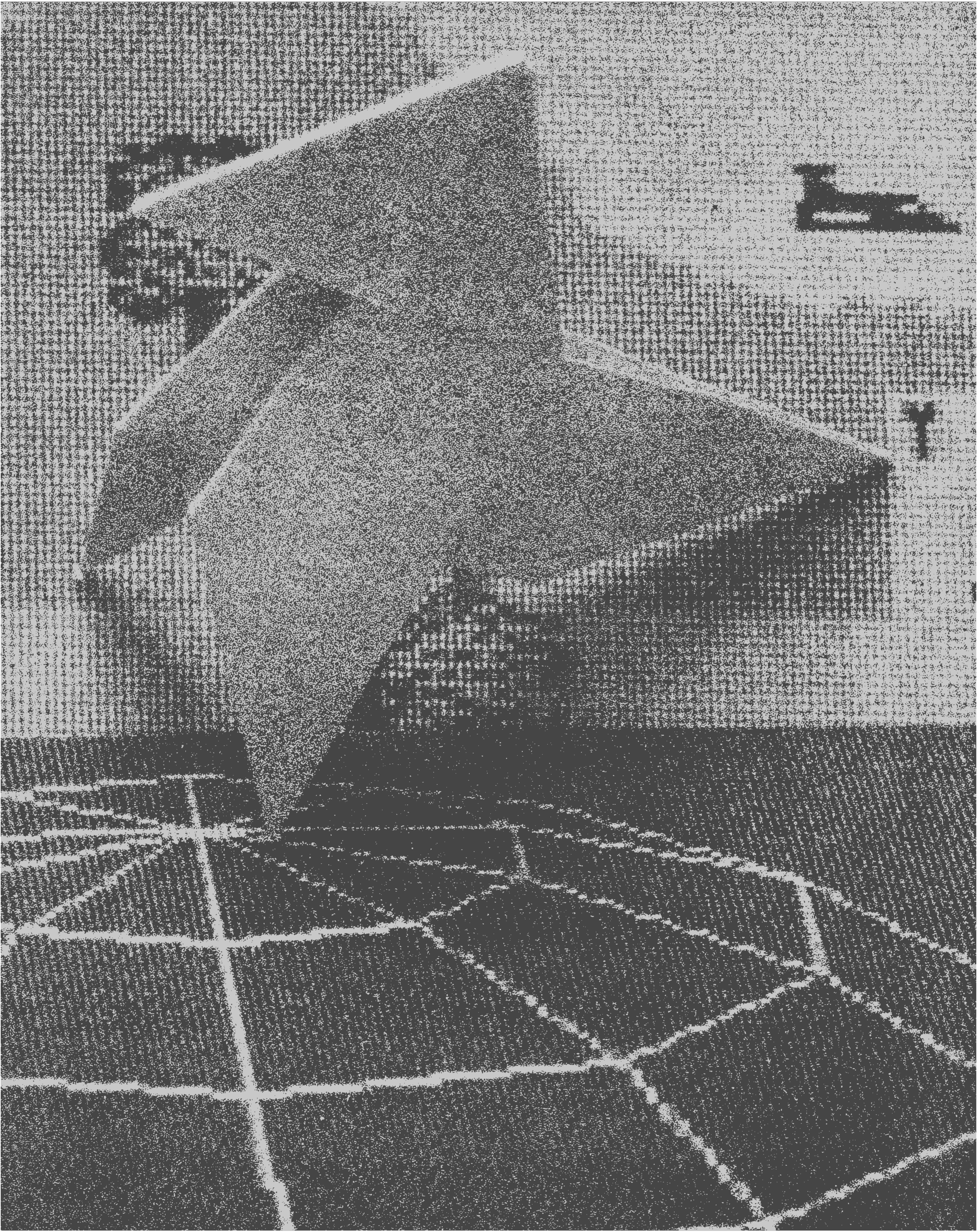
XII. Listado 1.12: no requiere datos.

XIII. Listado 1.13: no requiere datos.

XIV. Listado 1.14: no requiere datos.

XV. Listado 1.15: no requiere datos.

XVI. Listado 1.16: programa principal y subrutinas "fanfare" (soniquete), "worm" (gusano), "key" (tecla), "gobble" (engullido), "status" (estado) y "target" (objeto): utilice las teclas "I", "J", "K" y "M" para controlar el movimiento del gusano.



2 De coordenadas reales a pixels

Hemos visto que el Spectrum concibe su marco gráfico como una matriz rectangular de puntos direccionables o *pixels*. Dicha matriz está organizada en NXPIX (= 256) columnas verticales y NYPIX (= 176) filas horizontales. Cada uno de los *pixels* del conjunto de los NXPIX por NYPIX posibles, queda definido por un par de números enteros ordenados (I, J), donde queremos indicar el *pixel* situado en la I-ésima columna y en la J-ésima fila; naturalmente $0 \leqslant I \leqslant NXPIX - 1$ y $0 \leqslant J \leqslant NYPIX - 1$: el vector (0, 0) representa el *pixel* situado en la esquina inferior izquierda del papel. El BASIC del Spectrum dispone de varias instrucciones para el manejo de la matriz de *pixels;* la idea es considerarlos como puntos de luz que se pueden encender o apagar. Esto permite al operador aproximar líneas, o superficies, por un conjunto de puntos de color (los *pixels*).

El objetivo de los capítulos siguientes es dirigir al lector a la construcción de un paquete de programas gráfico (en 2 y 3 dimensiones) para el Sinclair Spectrum: los programas están escritos en BASIC y (salvo pocas excepciones) se basan en un número reducido de subrutinas primitivas que serán el objeto de este capítulo.

Primitivas que trasladan el espacio continuo al marco gráfico

En general, los gráficos realizados con ordenador representan puntos, líneas, superficies y volúmenes en espacios euclídeos continuos de dos y tres dimensiones; por el contrario, no disponemos de infinitos *pixels* infinitamente pequeños que se puedan asimilar a los puntos del espacio. Por otra parte, la definición de un objeto por un conjunto de pares de números discretos no es de mucha utilidad práctica. Necesitamos, por tanto, algún sistema que nos permita dibujar las vistas de los objetos en una

pantalla gráfica, donde medimos las posiciones en unidades reales: centímetros, kilómetros (¡o incluso años luz!). Nuestro foco de atención será, principalmente, realcionar el espacio real de dos dimensiones con la pantalla de *pixels*. Para ello empezaremos considerando, en primer lugar, la representación del espacio de dos dimensiones por medio de coordenadas cartesianas.

Podemos imaginarnos el espacio bidimensional como el plano de esta página extendido al infinito en todas direcciones. Empezaremos por elegir un punto fijo del plano, al que llamaremos origen de coordenadas. A continuación, trazamos por dicho punto una línea recta que se extiende por ambos extremos hacia el infinito; este es el *eje x*. Es aceptado tácitamente el colocar esta línea en posición horizontal. Una vez establecido el eje *x*, trazamos otra línea recta perpendicular a la anterior y que la corta en el origen, el *eje y;* siguiendo el mismo convenio, esta línea estará colocada verticalmente. Acto seguido, marcamos una escala a lo largo de cada eje: la unidad de distancia no precisa ser la misma sobre ambos, pero normalmente se suele tomar así (figura 2.1). Asumiremos también que los valores del eje *x* son positivos a la derecha del origen y negativos a su izquierda; de forma similar para el eje *y* consideraremos positivos los que estén situados por encima del origen y negativos los que estén por debajo.

Cualquier punto *p* del espacio queda definido unívocamente por medio de sus *coordenadas* (figura 2.1). La *coordenada x* (representada como X) es la distancia, medida sobre el eje horizontal, a la cual una línea perpendicular al eje *x* que pasa por el punto *p* corta a dicho eje. De una forma similar se define la *coordenada y* (Y). Estos dos valores, denominados *par de coordenadas* o *vector de dimensión dos,* se suelen escribir entre paréntesis, ordenados y separados por una coma: (X, Y). Es importante el orden en que se colocan ambas coordenadas, primero la coordenada *x* y luego la *y*.

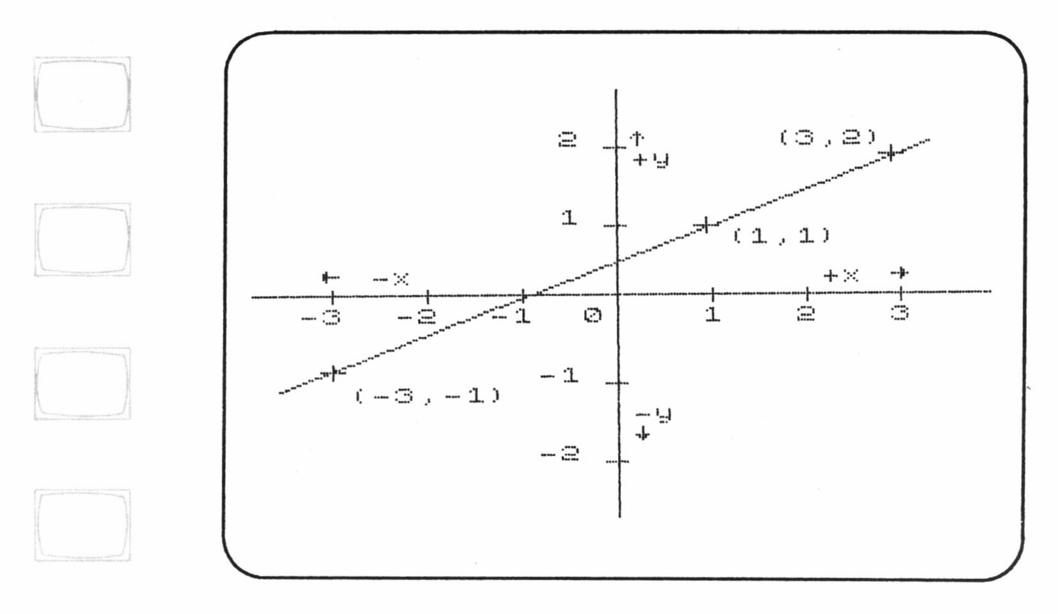


Figura 2.1

Frecuentemente denominaremos vector a un par de números ordenados. Por otra parte, un vector puede definir, también, una dirección en el plano: aquella que es paralela a la recta que pasa por el origen y por el punto (X, Y) —pero hablaremos con más detenimiento de esto (y de otras muchas cosas como límites y polígonos) en el capítulo 3.

Estamos ya en condiciones de construir las herramientas (las ya mencionadas subrutinas primitivas) que nos permitan trasladar estos conceptos geométricos a la matriz rectangular discreta de *pixels* que constituyen el marco gráfico.

Por el momento, centraremos nuestra atención en el espacio de dos dimensiones: a partir del capítulo 7 extenderemos nuestro territorio de caza a la tercera dimensión. En ambos casos, precisamos de un método para representar una superficie rectangular del espacio cartesiano bidimensional en el marco gráfico. Por simplificar, consideraremos que esta superficie tiene sus bordes paralelos a los ejes cartesianos *x* e *y*. Inicialmente, supondremos que el extremo inferior izquierdo del rectángulo a representar se encuentra sobre el origen de coordenadas (0.0, 0.0), siendo, además, su base de longitud HORIZ y su altura VERT. Identificamos, en primer lugar, el origen con el *pixel* (0, 0) y a continuación normalizamos los lados del rectángulo, de forma que ajuste exactamente en el marco gráfico; evidentemente, este ajuste solamente podrá realizarse si los cocientes HORIZ:VERT y NXPIX:NYPIX son iguales (esto es, 256:176). Solamente en contadas ocasiones, como se podrá suponer, esta proporción existe, y por tanto, definiremos un factor de escala, XYSCALE, que nos coloque el punto (HORIZ, VERT), del espacio real, bien sobre el borde superior, bien sobre el borde derecho del marco gráfico. Podemos considerar este rectángulo como una ventana sobre el espacio cartesiano; si liberamos a esta ventana del origen de coordenadas y, manteniendo sus bordes paralelos a los ejes, le permitimos moverse por todo el plano, podremos ver cualquier zona del espacio cartesiano. Como regla general, HORIZ y VERT estarán en proporción 3 a 2 aproximadamente.

Como indicamos, permitiremos al origen de coordenadas moverse de su posición inicial en el extremo inferior izquierdo del marco. Su posición relativa al origen inicial se guardará en las variables XORIG e YORIG, las componentes *x* e *y* respectivamente; inicialmente (XORIG, YORIG), tiene el valor (0.0, 0.0). Por tanto, un punto del espacio cartesiano de coordenadas (XPT, YPT), donde XPT e YPT son números reales, viene representado por el *pixel* cuya coordenada horizontal es INT ((XORIG + + XPT)*XYSCALE + 0.5) y cuya coordenada vertical es, similarmente, INT ((YORIG + YPT)*XYSCALE + 0.5); (INT es la función del BASIC que trunca la parte decimal de un número real y nos devuelve su parte entera). Ambas componentes son calculadas por las funciones FN X y FN Y (véase listado 2.2). A lo largo del trazado de una figura, debemos considerar la existencia de una "pluma de gráficos" que en realidad estará constituida por un par de números enteros, desplazándose por el marco gráfico; al comenzar la ejecución del dibujo estará situada en el *pixel* (0,0) y su posición, en cualquier instante, será el *pixel* (XPEN, YPEN). Las constantes NXPIX y NYPIX, y las variables XYSCALE, XPEN, YPEN, XORIG e YORIG, deben ser accesibles por cualquiera de las subrutinas que siguen; por tanto estos nombres serán reservados y no podrán emplearse para otras variables. Estas subrutinas fueron diseñadas específicamente para el Spectrum; de todas formas, comentaremos las ideas básicas que permitan construir subrutinas similares para otros equipos.

Como primera medida, si empleamos otra máquina, deberemos cambiar NXPIX y NYPIX a los valores correspondientes de la nueva.

Nuestra primera subrutina "comienzo" inicializa las variables requeridas y preparada la pantalla para el dibujo. El listado 2.1 es una posible implementación de la subrutina "comienzo" para el Spectrum.

Listado 2.1

```
9700 REM comienzo
9701 REM Datos de entrada HORIZ,
VERT
9709 REM Datos de salida NXPIX,N
YPIX,XORIG,YORIG,XYSCALE,XPEN,YP
EN
9710 LET XORIG=0: LET YORIG=0
9720 LET XPEN=0: LET YPEN=0
9730 LET NXPIX=256: LET NYPIX=17
6
9740 LET XYSCALE=NXPIX/HORIZ: LE
T YSCALE=NYPIX/VERT
9750 IF XYSCALE>YSCALE THEN  LET
 XYSCALE=YSCALE
9760 RETURN
```

Esta subrutina podría extenderse con la introducción del color; para ello, introducimos dos nuevas variables COLPAP y COLINK (enteras en el rango 0 a 7) para el color del papel y de la tinta respectivamente. Es necesario introducir la siguiente sentencia:

9725 PAPER COLPAP : INK COLINK

Si necesitáramos escribir esta subrutina para un microordenador distinto, todo lo que necesitaríamos hacer sería cambiar las instrucciones gráficas del BASIC del Spectrum por las correspondientes de la nueva máquina. La mayoría de las restantes subrutinas de este libro son independientes de la estructura del ordenador.

En "comienzo" y en muchas otras que veremos después, es necesario transformar las coordenadas *x* e *y* de un punto en su *pixel* equivalente, de forma que introducimos dos funciones FX y FY en el listado 2.2.

La siguiente subrutina primitiva (listado 2.3) es "fijaorigen", que nos permite trasladar el origen de coordenadas una distancia YMOVE en dirección horizontal e YMOVE en vertical (distancias medidas en la escala del sistema de coordenadas), corrigiendo el vector (XORIG, YORIG) a sus valores correspondientes. Después de

su ejecución, la pluma de dibujo se habrá desplazado al *pixel* correspondiente al nuevo origen.

Listado 2.2

```
9650 DEF FN X(Z)=INT ((XORIG+Z)*
XYSCALE+0.5)
9660 DEF FN Y(Z)=INT ((YORIG+Z)*
XYSCALE+0.5)
```

Listado 2.3

```
9600 REM fijaorigen
9601 REM Datos de entrada XORIG,
YORIG,XMOVE,YMOVE
9609 REM Datos de salida XORIG,Y
ORIG,XPEN,YPEN
9610 LET XORIG=XORIG+XMOVE: LET
YORIG=YORIG+YMOVE
9620 LET XPEN=FN X(0)
9630 LET YPEN=FN Y(0)
9640 RETURN
```

Con el fin de poder dibujar líneas rectas, introduciremos dos nuevas subrutinas: "trazapuntos" y "trazalíneas". La primera se encargará de llevar la pluma al *pixel* correspondiente al punto (en el espacio de coordenadas) donde comienza la línea; la segunda se ocupará de dibujar el segmento moviendo la pluma desde su posición presente (definida por una llamada previa a "fijaorigen", "trazapuntos" o "trazalíneas") hasta el *pixel* correspondiente al punto donde acaba la línea. Los listados 2.4 y 2.5 corresponden a las subrutinas "trazapuntos" y "trazalíneas" (diseñadas específicamente para el Spectrum). La subrutina "trazalíneas" contiene algunas sentencias que inicializan instrucciones que son propias del BASIC-Spectrum para el trazado de rectas (mientras que PLOT trabaja sobre valores absolutos, DRAW maneja valores relativos); por el contrario, la subrutina "trazapuntos" es independiente de la máquina. Si desea implementar estas subrutinas en otro microordenador, sólo necesitará cambiar la subrutina "trazalíneas".

Prácticamente todas las máquinas, salvo las más elementales, permiten almacenar estas subrutinas gráficas o sus equivalentes (y muchas más cuando el nivel de conocimiento aumenta) en un fichero de librería o en un dispositivo de memoria masiva, con lo que no será necesario teclearlas explícitamente en cada nuevo programa.

Listado 2.4

```
9500 REM trazapuntos
9501 REM Datos de entrada XPT,YP
T
9509 REM Datos de salida XPEN,YP
EN
9510 LET XPEN=FN X(XPT)
9520 LET YPEN=FN Y(YPT)
9530 RETURN
```

Listado 2.5

```
9400 REM recta
9401 REM Datos de entrada XPT,YP
T,XPEN,YPEN
9402 REM Datos de salida XPEN,YP
EN
9410 LET NXPEN=FN X(XPT)
9420 LET NYPEN=FN Y(YPT)
9430 PLOT XPEN,YPEN
9440 DRAW NXPEN-XPEN,NYPEN-YPEN
9450 LET XPEN=NXPEN: LET YPEN=NY
PEN
9460 RETURN
```

En el caso del Spectrum, podemos guardarlas como ficheros en cassettes de audio (y añadirlas al programa con la instrucción MERGE cuando las necesitemos). En el cassette que se acompaña con este libro encontrará estas subrutinas incluidas en la librería "rut1".

Ejemplo 2.1

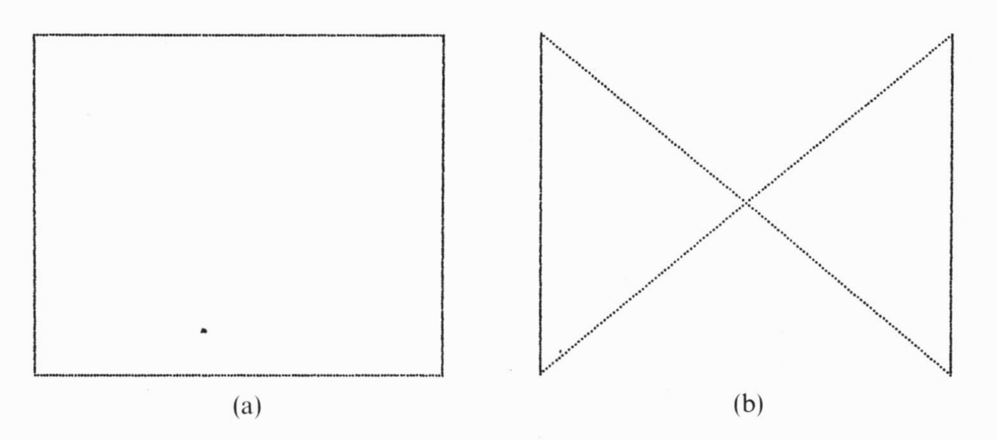
Identifique un rectángulo en el espacio cartesiano, de tamaño 30 × 20 unidades arbitrarias, con el marco gráfico del Spectrum. A continuación dibuje un cuadrado de 15 unidades de lado, centrado en el rectángulo (figura 2.2a).

Para centrar el cuadrado, ponemos el origen en (15.0, 10.0), y definimos las esquinas del cuadrado por los puntos (±7,5, ±7.5). Véase el listado 2.6.

Un punto importante a destacar es el orden en que se unen los vértices. Si intercambiamos, por ejemplo, las coordenadas de las esquinas segunda y tercera del cuadrado, la figura resultante sería la 2.2b en lugar de la 2.2a como se deseaba.

Listado 2.6

```
 100 REM dibujando un cuadrado
 109 REM Fija los identificadore
s de las rutinas graficas
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400
 120 LET HORIZ=30: LET VERT=20
 130 GO SUB start
 140 LET XMOVE=HORIZ*.5: LET YMO
VE=VERT*.5
 150 GO SUB setorigin
 159 REM Une las esquinas del cu
adrado en orden
 160 LET XPT=7.5: LET YPT=7.5: G
O SUB moveto
 170 LET XPT=-7.5: LET YPT=7.5:
GO SUB lineto
 180 LET XPT=-7.5: LET YPT=-7.5:
 GO SUB lineto
 190 LET XPT=7.5: LET YPT=-7.5:
GO SUB lineto
 200 LET XPT=7.5: LET YPT=7.5: G
O SUB lineto
 210 STOP
```



(a) (b)

Figura 2.2

A continuación escribiremos una subrutina primitiva a la que denominaremos "polígono" que emplea la instrucción del Spectrum para dibujar rectas. Su objetivo será el trazado de polígonos. Para su ejecución, deberemos proporcionarle los NPOL vértices del polígono en las variables X e Y (coordenadas *x* e *y* de cada uno de los puntos, habiendo definido X e Y como vectores de dimensión NPOL). También se incluye un programa principal de ejemplo que llama a esta subrutina (listado 2.7).

Listado 2.7

```
 100 REM programa principal/ lla
ma subrutina poligono
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400: LET polygon=300
 120 LET HORIZ=30: LET VERT=20
 130 GO SUB start
 140 LET XMOVE=HORIZ*.5: LET YMO
VE=VERT*.5
 150 GO SUB setorigin
 159 REM Lee los vertices del po
ligono
 160 DIM X(50): DIM Y(50)
 170 INPUT "TECLEA EL NUMERO DE
VERTICES ",NPOL
 180 FOR I=1 TO NPOL
 190 INPUT ("X("+STR$ I+") ");X(
I),("Y("+STR$ I+") ");Y(I)
 200 NEXT I
 210 GO SUB polygon
 220 STOP

 300 REM poligono
 301 REM Datos de entrada NPOL,X
(),Y()
 310 LET XPT=X(NPOL): LET YPT=Y(
NPOL): GO SUB moveto
 319 REM Une los vertices del po
ligono en orden
 320 FOR I=1 TO NPOL
 330 LET XPT=X(I): LET YPT=Y(I):
 GO SUB lineto
 340 NEXT I
 350 RETURN
```

Ejercicio 2.1

Si empleamos un Spectrum, podemos dibujar gráficos con distintos colores. Pero antes de realizar el dibujo tenemos que definir el color mediante la instrucción INK. Escriba una subrutina "poncolor" con un parámetro entero COLINK que realice esta tarea.

Ejercicio 2.2

En todas las subrutinas anteriores, la escala del dibujo (XYSCALE) se fija una vez a un determinado valor y no se cambia, siendo igual para ambas coordenadas. No hay necesidad de seguir este procedimiento; escriba una subrutina "factor" que cambie la escala horizontal en FX y la vertical en FY. Naturalmente, tendremos que definir dos factores de escala en vez de uno (XSCALE e YSCALE); y también habremos de corregir las subrutinas "comienzo", "fijaorigen", "trazapuntos", "trazalíneas" (véase también el capítulo 6).

Ejercicio 2.3

No hay ninguna razón por la que los ejes x e y estén orientados horizontal y verticalmente. Es más, no tienen por qué ser perpendiculares uno al otro. Experimente un poco con estas ideas y estudie las modificaciones a realizar en cada una de las subrutinas gráficas: "comienzo", "trazapuntos", etc.

Ejemplo 2.2

CalComp es uno de los primeros (y más populares) paquetes de programas gráficos. Contiene un conjunto de subrutinas para trazar ejes coordenados, calcular las escalas de éstos, construcción de gráficos, etc. Todas las subrutinas de CalComp se basan en la denominada "plot" (no tiene nada que ver con la instrucción PLOT del Spectrum) cuyo objetivo es el trazado de rectas; "plot" emplea tres parámetros: dos de ellos reales XPT e YPT, que se corresponden con las coordenadas de un punto del espacio, y un entero MOVE que contiene información sobre el movimiento de la pluma gráfica. MOVE puede tomar los valores ± 2 ó ± 3. Esta subrutina puede sustituir por sí sola a nuestras "fijaorigen", "trazapuntos" y "trazalíneas". Si MOVE es negativo, el origen de coordenadas se traslada al punto (XPT, YPT) del anterior sistema coordenado ("fijaorigen"). Cuando el valor absoluto de MOVE es 3 la pluma gráfica se mueve sin dibujar ("trazapuntos"), y, por último, cuando ABS(MOVE) es 2, el movimiento se realiza trazando una recta en la pantalla ("trazalíneas").

Aunque CalComp en su conjunto es demasiado complejo para intentar su elaboración completa, podemos implementar la subrutina "plot" y emplearla en lugar de "fijaorigen", "trazapuntos" y "trazalíneas" como el resto de las subrutinas mencionadas en este capítulo —véase el listado 2.8.

Listado 2.8

```
9800 REM plot/CalComp
9801 REM ENTRADA :XPT, YPT, XPEN
, YPEN, XORIG, YORIG, MODE     .
9802 REM SALIDA :XPEN, YPEN, XOR
IG, YORIG
9810 LET NYPEN = FN X(XPT)
9820 LET NYPEN = FN Y(YPT)
9830 IF ABS (MODE) = 2 THEN  PLO
T XPEN,YPEN: DRAW NXPEN-XPEN,NYP
EN-YPEN
9840 LET XPEN = NXPEN: LET YPEN
= NYPEN
9850 IF MODE < 0 THEN  LET XORIG
 = XORIG + XPT:  LET YORIG = YOR
IG + YPT
9860 RETURN
```

Como demostración de lo que puede hacerse con estas subrutinas gráficas, dibujaremos algunos modelos sencillos. Puede haber alguien que piense que la construcción de modelos geométricos es una forma frívola de matar el tiempo. Antes al contrario, consideramos que es uno de los métodos más didácticos para la comprensión de las técnicas de dibujo realizado con ordenador. A menudo, los temas geométricos que parecen realizados con el auxilio de subrutinas muy complejas son, en realidad, el resultado de programas muy elementales. Como la construcción de estos programas se puede realizar en poco tiempo, el efecto sobre el principiante es una inyección de moral y le da una mayor confianza en este campo. Pero, además, hay que tener en cuenta que los temas geométricos se emplean cuando se desea atraer la atención sobre un determinado artículo, y así se colocan frecuentemente en las portadas de libros, folletos de propaganda, etc. Los modelos geométricos son una forma agradable de introducir algunos de los conceptos en los que se basa el dibujo con ordenador. El siguiente programa, por ejemplo, resalta el importante papel que desempeñan las funciones trigonométricas (seno y coseno) y la medición de ángulos en radianes. Recuérdese que π radianes son equivalentes a 180 grados sexagesimales.

Ejemplo 2.3

La figura 2.3, un diseño muy popular por cierto, se construye uniendo cada vértice de un polígono regular de N lados (un N-gono) con todos los restantes. N es menor o igual que 30.

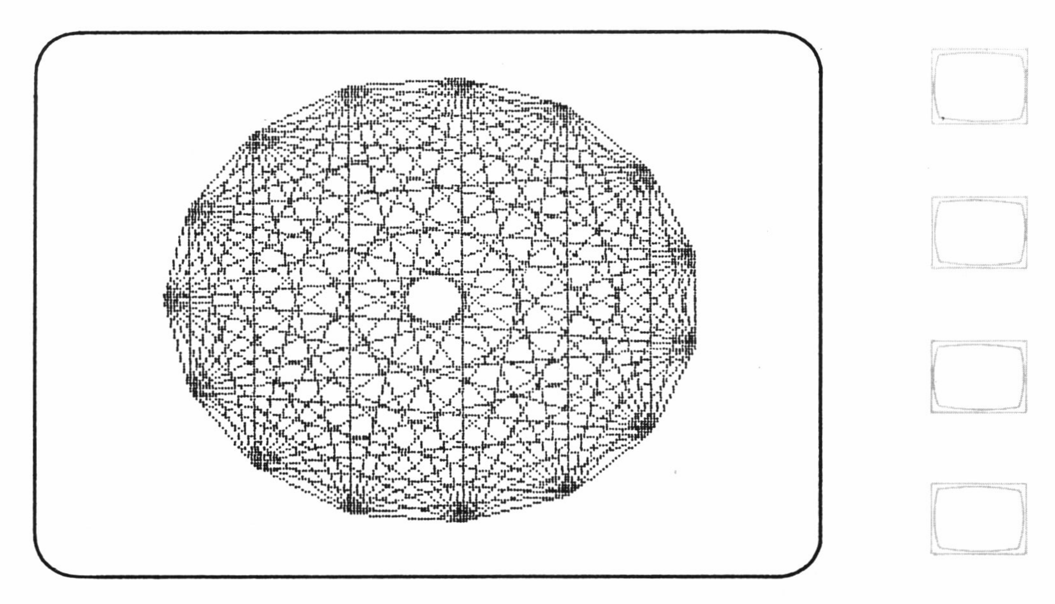


Figura 2.3

Colocamos el origen en el centro del dibujo, y todos los vértices sobre una circunferencia de radio unidad: los valores asignados a HORIZ y VERT (3, 2.1) se han escogido para que el gráfico presente un tamaño y proporción apropiados en la pantalla. Si uno de los vértices está sobre el eje x (el horizontal), entonces las coordenadas de todos ellos vienen dadas por la expresión: (COS (ALPHA), SIN (ALPHA), donde ALPHA es el ángulo $2\pi I/N$ e I toma los valores 1, 2, 3, ..., N. Por primera vez nos encontramos con una serie de puntos cuyas coordenadas son calculadas en lugar de introducidas por el teclado, como en el listado 2.6. Más aún, como el programa las emplea frecuentemente, parece lógico el calcularlas una sola vez y almacenarlas en vectores para su uso posterior. Obsérvese en el listado 2.9, que si $1 \leqslant I \leqslant J \leqslant N$, entonces el vértice J-ésimo no se une al I-ésimo, ya que el segmento ya ha sido trazado en el sentido opuesto.

Listado 2.9

```
 100 REM uniendo vertices de un
poligono regular de N lados

 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400: LET clip=8400
 120 LET HORIZ=3: LET VERT=2.1
 130 GO SUB start
 140 LET XMOVE=HORIZ*0.5: LET YM
```

```
OVE=VERT*0.5
 150 GO SUB setorigin
 160 DIM X(30): DIM Y(30)
 169 REM Fija los vertices del p
oligono regular en las matrices
X e Y                          .
 170 INPUT "TECLEA EL VALOR DE N
  ";N
 180 LET ALPHA=0: LET ADIF=2*PI/
N
 190 FOR I=1 TO N
 200 LET X(I)=COS ALPHA: LET Y(I
)=SIN ALPHA
 210 LET ALPHA=ALPHA+ADIF
 220 NEXT I
 229 REM Une el punto I al punto
 J. 1<=I<J<=N
 230 FOR I=1 TO N-1
 240 FOR J=I+1 TO N
 250 LET XPT=X(I): LET YPT=Y(I):
 GO SUB moveto
 260 LET XPT=X(J): LET YPT=Y(J):
 GO SUB lineto
 270 NEXT J
 280 NEXT I
 290 STOP
```

Se deben hacer dos puntualizaciones a la vista del ejemplo anterior: la primera trata sobre la *resolución.* Como el marco gráfico está constituido por una matriz discreta, las *líneas rectas* tienen que ser aproximadas por una sucesión de *pixels.* Por desgracia, la resolución del Spectrum, análogamente a la mayoría de los sistemas gráficos de microordenadores, es baja (es decir, NXPIX y NYPIX son algunos cientos) y las líneas aparecen dentadas; esto es cierto incluso para los sistemas gráficos de más alta resolución (como los terminales gráficos empleados en microfilmación), pero el tamaño del *pixel* es tan pequeño que las líneas aparecen como trazos continuos.

El segundo caso interesante se presenta cuando N es grande, entonces el polígono se aproxima a un círculo. Podemos desarrollar esta idea para escribir la subrutina "círculo1" (listado 2.10a), que dibuja un círculo de radio R con centro en (XCENT, YCENT) y cuya apariencia es similar a la figura 2.4. Nótese que los ángulos se miden en radianes; de forma que el incremento angular es 3/(R*XYSCALE); se ha elegido este valor, dependiente del radio, como solución de compromiso entre el tiempo de cálculo y aproximación al círculo geométrico. Como las coordenadas de los vértices no se necesitan nada más que una vez, no es necesario almacenarlas. De nuevo se evidencia la limitada resolución de la pantalla al observar la circunferencia terminada.

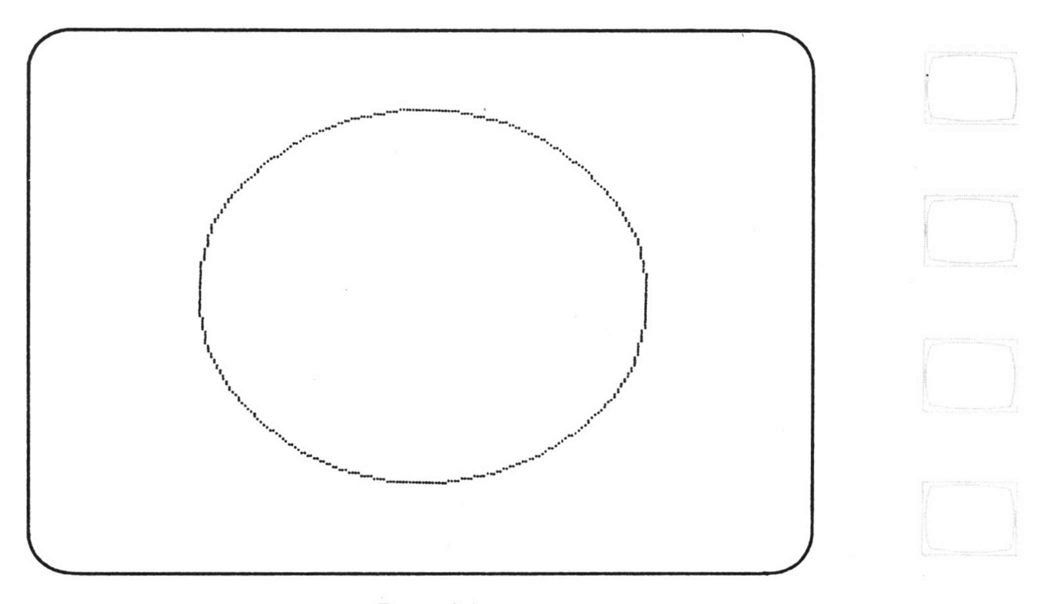


Figura 2.4

Listado 2.10a

```
 300 REM circulo1
 301 REM Datos de entrada XCENT,
YCENT,R,XYSCALE
 310 LET XMOVE=XCENT: LET YMOVE=
YCENT: GO SUB setorigin
 320 LET ADIF=3/(R*XYSCALE)
 330 LET XPT=R: LET YPT=0: GO SU
B moveto
 339 REM Calcula y une los punto
s (XPT,YPT) del circulo
 340 FOR A=ADIF  TO 2*PI STEP AD
IF
 350 LET XPT=R*COS A: LET YPT=R*
SIN A: GO SUB lineto
 360 NEXT A
 370 RETURN
```

Como vimos anteriormente, el Spectrum tiene una instrución BASIC (CIRCLE), que nos permité el trazado de un círculo. Así que la aprovecharemos en la subrutina primitiva "círculo2" (listado 2.10b) para el dibujo de círculos; evidentemente, es mucho más rápida y de mayor precisión que "círculo1".

Listado 2.10b

```
 400 REM circulo2
 401 REM Datos de entrada XCENT,
YCENT,R,XYSCALE
 410 CIRCLE FN X(XCENT),FN Y(YCE
NT),R*XYSCALE
 420 RETURN
```

Cuando usemos alguna subrutina, debemos estar prevenidos contra los llamados *efectos colaterales* que se pueden producir; por ejemplo, ¿cambia la subrutina el origen de coordenadas o la posición de la pluma? En el caso de la subrutina 2.10a, ambos puntos varían su posición, mientras que la correspondiente al listado 2.10b los deja en la misma posición en que estaban antes de su ejecución. Es conveniente por tanto la siguiente sentencia a la subrutina “círculo1”:

```
370 LET XMOVE = -XCENT : LET YMOVE = -YCENT :
    GO SUB setorigin : RETURN
```

Ejercicio 2.4

Escriba una subrutina que dibuje una elipse cuyo eje mayor mida A unidades (horizontal) y el eje menor B unidades (vertical). Tenga en cuenta que las coordenadas de un punto de la elipse son (A cos α, B sen α) donde $0 \leqslant \alpha \leqslant 2\pi$. Sin embargo, a diferencia del círculo, α no es el ángulo que forman el eje x y la recta que une el punto con el origen; es simplemente un parámetro descriptivo (y útil).

Incluya esta subrutina en un programa que dibuje un diagrama similar al de la figura 2.5. Hay dos puntos a destacar: 1) no es necesario que A sea mayor que B; y 2) observe la ilusión óptica de la presencia de dos diagonales blancas. En la figura 2.3 se presenta otra ilusión —unos círculos oscuros concéntricos. El estudio de las ilusiones ópticas es fascinante y es una fuente inagotable de ideas para modelos geométricos. Este ejercicio se introdujo porque abre el camino a la técnica general de trazado de curvas (capítulos 3 y 6).

Ejemplo 2.4

Como desarrollo de esta idea, el siguiente paso es la construcción de una espiral. De nuevo la forma general de la curva alrededor del origen es (R cos α, R sen α), pero ahora α varía entre los ángulos β y $\beta + 2N\pi$, donde β (el parámetro BETA) es el ángulo inicial que forma la normal a la espiral con la parte positiva del eje x, y N es el número de vueltas que describe la espiral alrededor del origen. La di-

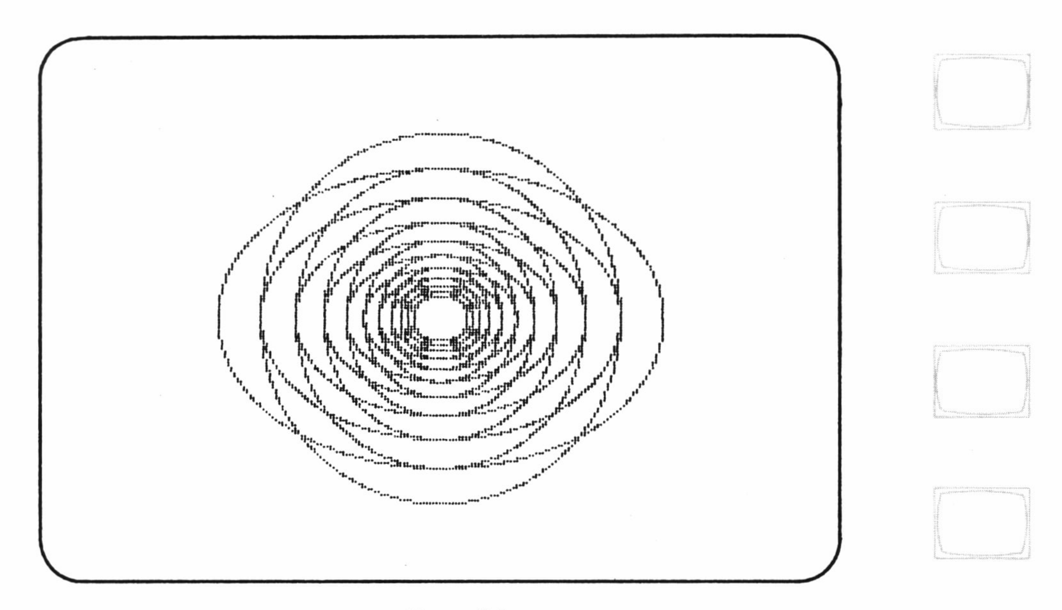


Figura 2.5

ferencia radica en que el radio R no es constante, sino que varía con el valor de α: si RMAX es el radio máximo de la espiral entonces R viene dado por la fórmula:

$$R = RMAX(\alpha - \beta)/2N\pi$$

Esta subrutina, que dibuja la espiral centrada en (XCENT, YCENT), no origina ningún efecto colateral, ya que devolveremos el origen de coordenadas a su anterior posición antes de terminar su ejecución.

Ejercicio 2.5

El listado 2.11a produce un diagrama similar a la figura 2.6a (con XCENT = = 0, YCENT = 0, N = 4, BETA = 1 y RMAX = 3). ¿Qué ocurriría si a RMAX le damos el valor −3? Utilice la subrutina en un programa que genere la figura 2.6c. Adviértase, de nuevo, la ilusión óptica que aparece cuando el observador mueve su cabeza en círculos frente al diagrama, manteniendo la dirección horizontal (y por consiguiente la vertical) paralela a la original. ¡Las espirales parecen girar alrededor del centro!

Listado 2.11a

```
 300 REM espiral1
 301 REM Datos de entrada XCENT,
YCENT,RMAX,N,BETA
 310 LET XMOVE=XCENT: LET YMOVE=
```

```
YCENT: GO SUB setorigin
 320 LET ADIF=PI/50: LET ALPHA=B
ETA
 330 LET RDIF=RMAX/(N*100)
 339 REM Calcula y une los punto
s (XPT,YPT) de la espiral
 340 FOR R=RDIF  TO RMAX STEP RD
IF
 350 LET XPT=R*COS ALPHA: LET YP
T=R*SIN ALPHA: GO SUB lineto
 360 LET ALPHA=ALPHA+ADIF
 370 NEXT R
 380 LET XMOVE=-XCENT: LET YMOVE
=-YCENT: GO SUB setorigin
 390 RETURN
```

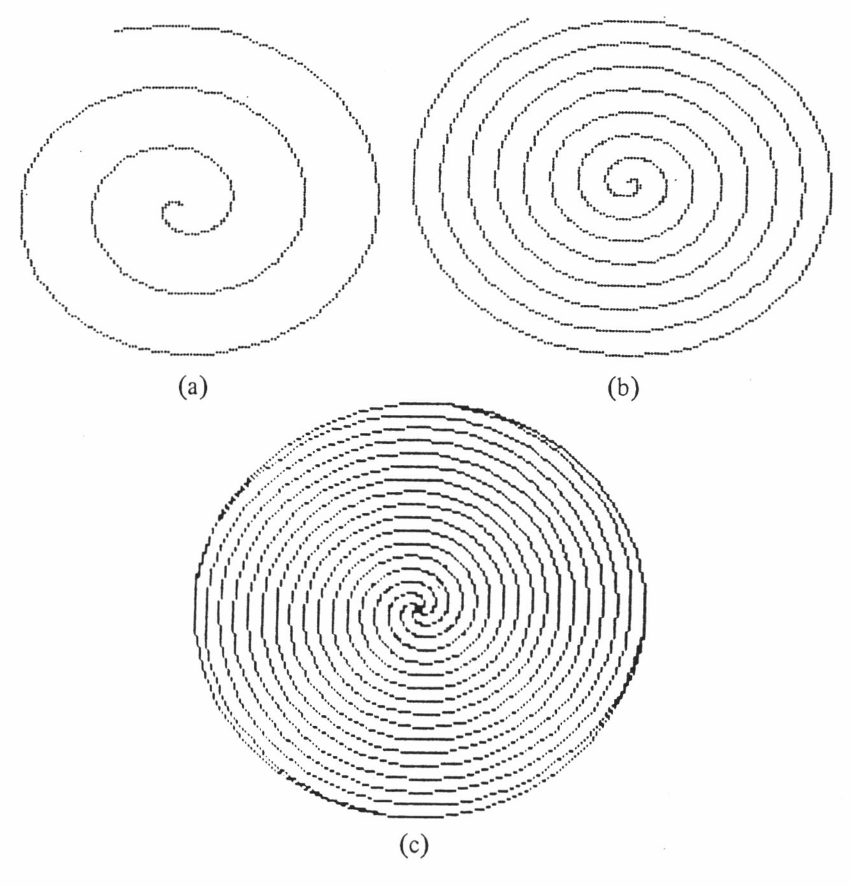
Ejemplo 2.5

Las espirales han sido usadas como motivo artístico o de decoración durante miles de años; sin embargo, la mayoría de las antiguas no eran verdaderas espirales, sino que consistían en varios semicírculos conectados. El listado 2.11b (una subrutina cuyos parámetros son RMAX, N y SIGN, la dirección de giro, que puede tomar como valores ± 1) nos permite dibujar espirales semicirculares por medio de la instrucción DRAW, y además es mucho más eficiente que el método del listado 2.11a, aunque este último es más preciso (como resultado de la subrutina 2.11b, véase la figura 2.6b).

Listado 2.11b

```
 300 REM espiral celtica
 301 REM Datos de entrada XCENT,
YCENT,RMAX,N,SIGN
 310 LET XMOVE=XCENT: LET YMOVE=
YCENT: GO SUB setorigin
 320 PLOT XPEN,YPEN
 330 LET R=0: LET RDIF=RMAX/(N*2
): LET S=1
 339 REM Construye la espiral us
ando DRAW para producir una seri
e de semicirculos
 340 FOR I=1 TO N*2
```

```
 350 LET R=R+RDIF: LET XPIX=S*R*
XYSCALE
 360 DRAW XPIX,0,SIGN*PI
 370 LET S=-S
 380 NEXT I
 390 RETURN
```



(a) (b)

(c)

Figura 2.6

Ejercicio 2.6

Siga la lógica del listado 2.6b, y extiéndalo de forma que la normal a la curva original no esté sobre el eje *x*, sino que forme un ángulo BETA con él. En la figura 1.10 hay un ejemplo de *triskele,* que se compone de varios tercios de circunferencia conectados. Experimente en la construcción de éstos y variaciones sobre este tema, tales como cuartos de círculo, etc.

Ejemplo 2.6

Escriba una subrutina (listado 2.12) que dibuje diagramas similares a la figura 2.7.

Aquí se presenta el concepto de *envolvente*. En vez de dibujar una curva por medio de una serie de pequeños trazos rectilíneos (como la circunferencia del listado 2.9), utilizamos una serie de rectas que son tangentes a la curva. Por ejemplo, la figura muestra cuatro hipérbolas rectangulares situadas en los *cuadrantes* del plano.

Tomamos N puntos en cada uno de los cuatro brazos (de longitud unitaria) que dividen el plano en los cuatro cuadrantes. Los 4N puntos son por tanto ($\pm$I/N, 0.0) y (0.0, $\pm$I/N) donde I = 1, 2, ..., N.

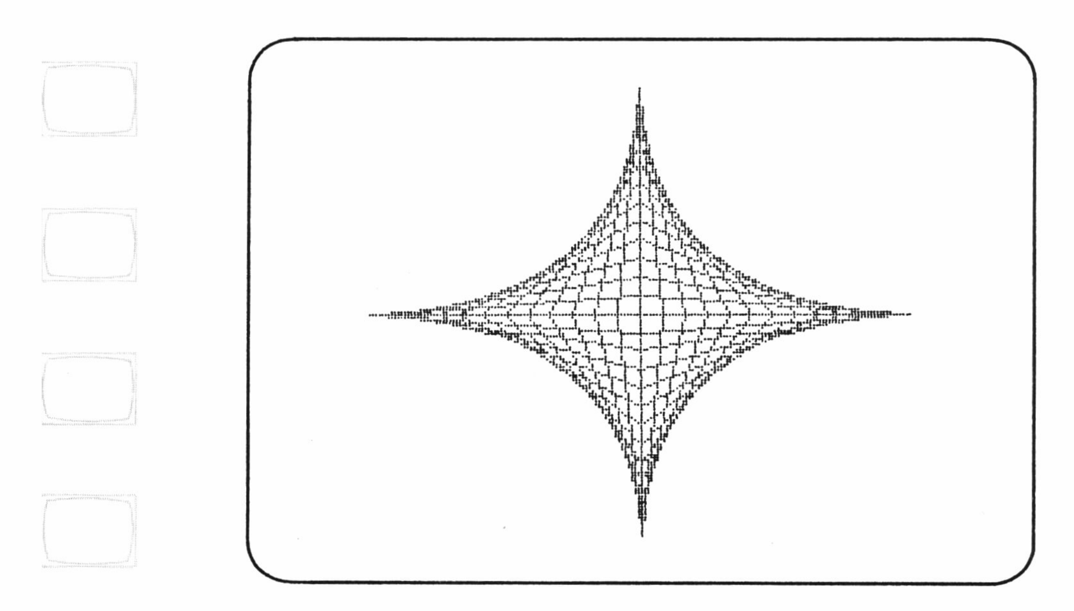


Figura 2.7

Listado 2.12

```
 100 REM programa principal
         (ejemplo de envolvente
)
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400
 120 LET HORIZ=3: LET VERT=2.1
 130 GO SUB start
 140 LET XMOVE=HORIZ*0.5: LET YM
OVE=VERT*0.5
```

```
 150 GO SUB setorigin
 159 REM Dibuja las escalas de l
os ejes en el area de graficos
 160 INPUT "TECLEA N ";N
 170 LET XPT= 1: LET YPT= 0: GO
SUB moveto
 180 LET XPT=-1: LET YPT= 0: GO
SUB lineto
 190 LET XPT= 0: LET YPT= 1: GO
SUB lineto
 200 LET XPT= 0: LET YPT=-1: GO
SUB lineto
 208 REM Produce N grupos de 4 p
untos, cada uno en un eje
 209 REM Une los puntos de cada
grupo por orden
 210 FOR I=1 TO N
 220 LET ID1=I/N: LET ID2=(N+1-I
)/N
 230 LET XPT= ID1: LET YPT=   0:
 GO SUB moveto
 240 LET XPT=   0: LET YPT= ID2:
 GO SUB lineto
 250 LET XPT=-ID1: LET YPT=   0:
 GO SUB lineto
 260 LET XPT=   0: LET YPT=-ID2:
 GO SUB lineto
 270 LET XPT= ID1: LET YPT=   0:
 GO SUB lineto
 280 NEXT I
 290 STOP
```

Ejercicio 2.7

Generalice este subprograma de forma que haya un número de brazos variable, M, saliendo del origen y dividiendo el plano en segmentos iguales.

Ejercicio 2.8

Dibuje un diagrama similar al de la figura 2.8; la subrutina tendrá un parámetro entero N. Calculará 4N puntos $\{P(I): I = 1, 2, ..., 4N\}$ sobre el perímetro de un cuadrado, de lado unidad, empezando en un vértice. Hay un punto en cada vértice, y la distancia entre los puntos consecutivos es 1/N. A continuación se unen por

medio de segmentos los pares de puntos según las siguiente regla: P(I) se une a P(J) si J – I (módulo 4N) pertenece a la secuencia 1, 1 + 2, 1 + 2 + 3, ... Por ejemplo, si N = 10 entonces P(20) se unirá a P(21), P(23), P(26), P(30), P(35), P(1), P(8) y P(16). Si se dibuja el cuadrado exterior, no es necesario dibujar los segmentos que unan pares de puntos sobre el mismo lado. Siguiendo el ejemplo anterior, P(20) es un vértice, así que nos podremos ahorrar los segmentos correspondientes a P(16), P(21), P(23), P(26) y P(30).

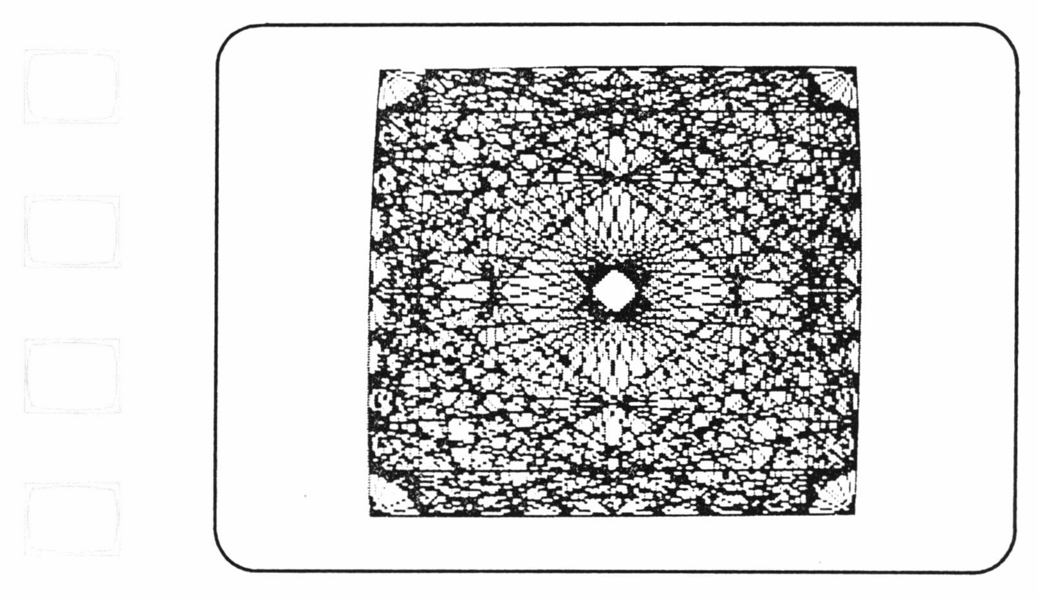


Figura 2.8

***Ejemplo* 2.7**

Sea un programa que imite a un espirógrafo, que produzca diagramas similares al de la figura 2.9.

Un espirógrafo consiste en un disco dentado que engrana sobre la parte interna de otra corona circular, la cual se coloca sobre un papel. Sean A y B dos números enteros correspondientes a los radios de la circunferencia interior de la corona y exterior del disco. El disco siempre está en contacto con la corona. A una distancia D (entera) del centro del disco hay un pequeño agujero en el cual se coloca un lápiz. Si se hace girar el disco de forma que ambas piezas estén siempre en contacto, el engranaje evita que haya deslizamiento. En su movimiento el lápiz dibujará una figura cerrada, ya que en algún momento volveremos a la posición original.

Inicialmente supondremos que el centro del disco y el agujero están sobre la parte positiva del eje x, siendo el origen de coordenadas el centro de la corona circular. Para imitar al espirógrafo, tenemos que determinar las coordenadas de un punto que esté sobre el trazo del lápiz. Sea α el ángulo que forman el eje x y la recta que

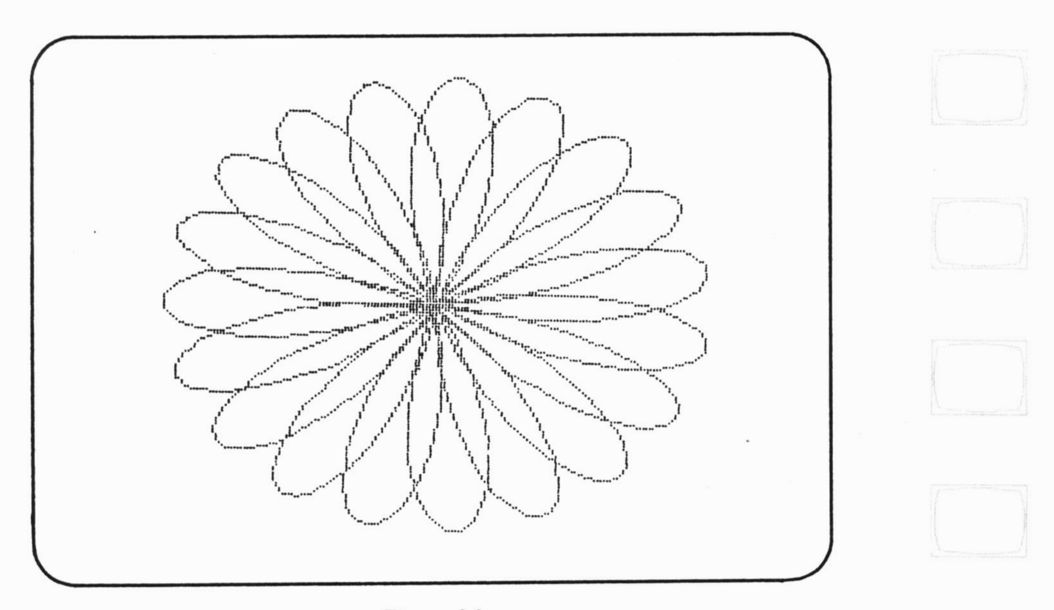


Figura 2.9

une el origen con el punto de contacto entre el disco y la corona. El punto de contacto será por tanto (A cos α, A sen α) y el centro del disco tendrá como coordenadas ((A − B) cos α, (A − B) sen α). Si llamamos β al ángulo formado por la recta que une el centro del disco con el agujero y el eje *x*, entonces las coordenadas del agujero serán:

$$((A - B) \cos \alpha + D \cos \beta, (A - B) \text{ sen } \alpha + D \text{ sen } \beta)$$

El punto de contacto entre el disco y la corona se habrá movido una distancia Aα alrededor de la corona, y una distancia $-B\beta$ alrededor del disco (el signo menos se debe a que α y β tienen orientaciones contrarias). Como el disco rueda sin deslizamiento, ambas distancias deberán ser iguales, y nos resulta la siguiente ecuación: $\beta = -(A/B)\alpha$. El lápiz volverá a su posición original cuando tanto α como β sean múltiplos de 2π. Cuando $\alpha = 2NX$, β tendrá el valor $-N(A/B)2\pi$: por tanto el lápiz regresa a su posición inicial por primera vez cuando N(A/B) sea un número entero por vez primera; o sea, cuando N sea igual a B dividido por el máximo común divisor de A y B. La subrutina "Euclides" (listado 2.13) emplea el algoritmo de Euclides para el cálculo del máximo común divisor (variable HCF del programa) de dos números naturales A y B.

Esta función se usa por la subrutina "espirógrafo" (listado 2.13), que calcula el valor de N y varía α (ALPHA) entre 0 y $2N\pi$ con incrementos de $\pi/100$; para cada α, calculamos el valor de β (BETA) y pintamos el punto correspondiente al agujero. La figura 2.9 fue dibujada por la subrutina "espirógrafo" con los siguientes valores: A = 12, B = 7 y D = 5. Los parámetros HORIZ y VERT deben elegirse de forma que el dibujo quepa dentro de la pantalla, en este ejemplo tomamos HORIZ = = 30 y VERT = 20.

Listado 2.13

```
 200 REM Euclides
 201 REM Datos de entrada A,B
 202 REM Datos de salida Maximo
comun divisor
 210 LET I=A: LET HCF=B
 220 IF A<B THEN  LET I=B: LET H
CF=A
 230 LET J=I-INT (I/HCF)*HCF
 240 IF J=0 THEN  RETURN
 250 LET I=HCF: LET HCF=J: GO TO
 230
 260 LET XPT=   0: LET YPT=-ID2:
 GO SUB lineto
 270 LET XPT= ID1: LET YPT=   0:
 GO SUB lineto
 280 NEXT I
 290 STOP

 300 REM espirografo
 301 REM Datos de entrada A,B,D
 310 LET RAB=A-B: LET ALPHA=0: L
ET ADIF=PI/50: LET AOB=A/B
 320 GO SUB Euclid: LET N=B/HCF:
 LET NO=100*N
 330 LET XPT=RAB+D: LET YPT=0: G
O SUB moveto
 339 REM Calcula y une los punto
s de la espiral
 340 FOR I=1 TO NO
 350 LET ALPHA=ALPHA+ADIF
 360 LET BETA=ALPHA*AOB
 370 LET XPT=RAB*COS ALPHA+D*COS
 BETA
 380 LET YPT=RAB*SIN ALPHA-D*SIN
 BETA
 390 GO SUB lineto
 400 NEXT I
 410 RETURN
```

Este ejemplo demuestra que el trazado de modelos geométricos no es tan sencillo como aparenta. Incluso una figura tan sencilla como la de la figura 2.8 precisa de la ayuda de Euclides. Conforme nos adentremos en el territorio de las técnicas de dibujo con ordenador, descubriremos que necesitamos, como mínimo, algunos conocimientos básicos de cálculo, álgebra, geometría euclídea y teoría de números, y no sólo de geometría cartesiana como se puede (erróneamente) suponer. Prepárese a "saquear" su biblioteca más próxima (o a amargarle la vida a su amable vecino matemático) en busca de la información necesaria.

Programas completos

En este punto agruparemos los listados 2.1, "comienzo" *(start)*; 2.2 "dos funciones FN X y FN Y"; 2.3, "fijaorigen" *(setorigin)*; 2.4, "trazapuntos" *(moveto)*; y 2.5, "trazalíneas" *(lineto)* bajo el nombre de "rut1". Más adelante cambiaremos el listado 2.5 por el 3.3, "recorte" *(clip)* y una nueva versión de "trazalíneas" *(lineto)*.

I. "rut1" y el listado 2.6, "dibujando un cuadrado" *(drawing a square)*: sin datos de entrada.

II. "rut1" y el listado 2.7, "programa principal" *(main program)* y "polígono" *(poligon)*: precisan el número de lados del polígono, y las coordenadas de los vértices X/Y ($-15 \leqslant X \leqslant 15$ y $-10 \leqslant Y \leqslant 10$).

III. "rut1" y el listado 2.9, "uniendo vértices de un polígono regular de N lados" *(joining vertices of regular N-gon)*: necesita un entero $N \leqslant 30$.

IV. "rut1" y su propio "programa principal" *(main program)* (el listado 2.7 puede ser útil como modelo): llama a los listados 2.10a, "círculo1" *(circle1)* y 2.10b "círculo2" *(circle2)*. Cada subrutina precisa el centro (XCENT, YCENT) y el radio R. Seleccione estos valores de forma que el gráfico sea consistente con sus valores de HORIZ, VERT, XMOVE e YMOVE. Puede ensayar con 30, 20, 15 y 10, respectivamente. Como ejemplo llame a "círculo1" con centro en (1, −1) y radio 8, y a "círculo 2" con centro en (1, 2) y radio 5.

V. "rut1" y su propio "programa principal" que llame a los listados 2.11a "espiral1" *(spiral1)* y 2.11b "espiral céltica" *(celtic)*. Cada una de las subrutinas requiere el centro (XCENT, YCENT), el radio máximo RMAX y el número de vueltas de la espiral N. El listado 2.11a "espiral1" también precisa un ángulo BETA, mientras que 2.11b "espiral céltica" necesita un valor SIGN, que es ± 1. Seleccione estos valores de forma que la figura sea consistente con sus valores de HORIZ, VERT, XMOVE e YMOVE (por ejemplo: 30, 20, 15 y 10). Puede llamar a "espiral1" con centro en (−1, 1), RMAX = 8, N = 3 y BETA = = 2, y a "espiral céltica" con centro en (1, 2), RMAX = 5 y SIGN = −1; inténtelo también con SIGN = = + 1.

VI. "rut1" y el listado 2.12, "envolvente" *(envelope):* precisa un entero N, $2 \leqslant N \leqslant 30$.

VII. "rut1" y el listado 2.13 "Euclides" *(Euclid)* y "espirógrafo" *(spiro):* requiere tres enteros A, B y D, donde $A > B > D$. Elija HORIZ, VERT, etc., de forma que el dibujo quepa en la pantalla: tome XMOVE = HORIZ*0.5, YMOVE = 0.5*VERT (para "fijaorigen"), donde tanto HORIZ como VERT son mayores que $2*(A - B + D)$.

3
Geometría cartesiana en dos dimensiones

En el capítulo 2 presentamos el concepto de sistema de ejes coordenados en dos dimensiones; allí definimos los puntos del espacio en forma de vectores, que nos permitían trazar segmentos rectilíneos entre pares de puntos. En buena ley, una *línea recta* en un espacio bidimensional no es un segmento de longitud finita, sino que se extiende hacia el infinito en ambos sentidos, de forma que se nos presenta la necesidad de representar un punto cualquiera de dicha recta.

Se nos ha enseñado que la ecuación de una recta es $y = mx + c$, es decir, la relación entre las coordenadas x e y de un punto perteneciente a la recta, donde m es la tangente trigonométrica del ángulo que forma la recta con la parte positiva del eje x, y c es el punto de intersección de la recta con el eje y; o sea, cuando $x = 0$, $y = c$. Esta fórmula será muy conocida, pero no es muy útil. ¿Qué ocurre cuando la recta es vertical? ¡m es infinito! Una fórmula de mayor utilidad es

$$ay = bx + c$$

Esta nueva ecuación puede aplicarse a todas las rectas del plano: si es vertical, a se anula; (b/a) es ahora la tangente del ángulo que la recta forma con el semieje x positivo, y el punto de corte con el eje y es (c/a) supuesto que a sea distinto de 0. Similarmente, el punto de corte con el eje x es $(-c/b)$ siempre que b no sea 0. La recta es paralela al eje y si a es 0 y al eje x si la que se anula es b.

En las siguientes páginas emplearemos frecuentemente esta ecuación; sin embargo, vamos a proponer a continuación otro método para definir una recta, que quizá sea más fructífero. Antes de exponer este nuevo método tenemos que definir dos operaciones sobre vectores (la multiplicación por un escalar y la suma de vectores) y des-

cribir otra operación necesaria —el valor absoluto (módulo) de un vector. Sean dos vectores $\boldsymbol{p}_1 \equiv (x_1, y_1)$ y $\boldsymbol{p}_2 \equiv (x_2, y_2)$; definimos las siguientes operaciones:

Multiplicación por un escalar: $k\boldsymbol{p}_1 = (k \times x_1, k \times y_1)$, multiplicamos cada una de las coordenadas por un valor escalar (es decir, un número real) k.

Suma de vectores: $\boldsymbol{p}_1 + \boldsymbol{p}_2 = (x_1 + x_2, y_1 + y_2)$, se suman por una parte las coordenadas x y por otra las y.

Valor absoluto: $|\boldsymbol{p}_1| = \sqrt{(x_1^2 + y_1^2)}$ es la distancia entre el punto $\boldsymbol{p}_1$ y el origen (a veces se le denomina amplitud, longitud o módulo del vector).

Para definir una línea recta elegimos, en primer lugar, un par de puntos de la recta, de nuevo los llamaremos $\boldsymbol{p}_1 \equiv (x_1, y_1)$ y $\boldsymbol{p}_2 \equiv (x_2, y_2)$. Cualquier punto $\boldsymbol{p}(\mu) \equiv$ $\equiv (x, y)$ de la recta está dado por la combinación de multiplicaciones por escalares y suma de vectores:

$$(1 - \mu)\boldsymbol{p}_1 + \mu\boldsymbol{p}_2 \text{ para algún valor de } \mu \text{ real}$$

que en forma desarrollada nos da el vector $((1 - \mu) \times x_1 + \mu \times x_2, (1 - \mu) \times y_1 + \mu \times$ $\times y_2)$. Hemos colocado μ entre paréntesis detrás de $\boldsymbol{p}$ para indicar que el vector $\boldsymbol{p}$ depende de valor que tome μ. Más adelante, cuando hayamos asimilado totalmente la dependencia de $\boldsymbol{p}$ con μ, quitaremos el paréntesis (μ). Si $0 \leqslant \mu \leqslant 1$, entonces $\boldsymbol{p}(\mu)$ se encuentra entre los puntos $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$. Para un punto dado $\boldsymbol{p}(\mu)$, el valor de μ viene dado por la relación

$$\mu = \frac{\text{distancia de } \boldsymbol{p}(\mu) \text{ a } \boldsymbol{p}_1}{\text{distancia de } \boldsymbol{p}_2 \text{ a } \boldsymbol{p}_1}$$

donde la distancia se toma positiva si $\boldsymbol{p}(\mu)$ está en el mismo lado de $\boldsymbol{p}_1$ que $\boldsymbol{p}_2$, y negativa en el caso contrario. La distancia entre dos puntos dados por sus vectores $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$ puede calcularse mediante el teorema de Pitágoras:

$$|\boldsymbol{p}_2 - \boldsymbol{p}_1| = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$$

En la figura 2.1 tenemos un segmento entre los puntos $(-3, -1) \equiv \boldsymbol{p}(0)$ y $(3, 2) \equiv \boldsymbol{p}(1)$: el punto $(1, 1)$, que pertenece a la recta es $\boldsymbol{p}(2/3)$. Obsérvese que la distancia entre $(-3, -1)$ y $(3, 2)$ es $3\sqrt{5}$, mientras que la que separa a $(-3, -1)$ de $(1, 1)$ es $2\sqrt{5}$. A partir de ahora omitiremos la dependencia explícita del vector $\boldsymbol{p}$ con μ (representaremos a $\boldsymbol{p}(\mu)$ simplemente por $\boldsymbol{p}$).

Ejemplo 3.1

Podemos ilustrar estas ideas por medio del trazado del modelo de la figura 3.1. A primera vista parece complicado, pero en una inspección más detallada se puede descomponer en un cuadrado, circunscrito a otro cuadrado, circunscrito a otro cuadrado, etc. Los cuadrados se van haciendo cada vez más pequeños y además rotan un ángulo constante sobre el anterior. Con el fin de dibujar el diagrama, necesita-

mos de una técnica que, partiendo de un cuadrado general, nos dibuje otro cuadrado más pequeño inscrito en el anterior y rotado un cierto ángulo fijo. Sean $\{(x_i, y_i)|i = 1, 2, 3, 4\}$ los cuatro vértices del cuadrado de partida, cuyo lado i-ésimo será el segmento que une los puntos (x_i, y_i) y $(x_i + 1, y_i + 1)$, donde la suma se realiza para los subíndices de forma cíclica (es decir, $4 + 1 = 1$). Un punto cualquiera del cuadrado (x'_i, y'_i) situado sobre este lado será el:

$$((1 - \mu) \times x_i + \mu \times x_{i+1}, (1 - \mu) \times y_i + \mu \times y_{i+1}) \text{ donde } 0 \leqslant \mu \leqslant 1$$

Incidentalmente, $\mu : 1 - \mu$ es la proporción en la que dividimos el lado del cuadrado. Si μ es constante y los cuatro puntos $\{(x'_i, y'_i)|i = 1, 2, 3, 4\}$ se calculan con la expresión anterior, entonces los lados del nuevo cuadrado formarán un ángulo $\alpha = \text{arc tan} [\mu/(1 - \mu)]$ con el lado correspondiente del cuadrado exterior. Para que el ángulo de giro permanezca constante, lo único que tenemos que hacer es mantener el mismo valor de μ. En el listado 3.1, que genera la figura 3.1, hay 21 cuadrados y μ vale 0.1.

Listado 3.1

```
 100 REM programa principal
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400: LET clip=8400
 120 LET HORIZ=3: LET VERT=2.1
 130 GO SUB start
 140 LET XMOVE=HORIZ*0.5: LET YM
OVE=VERT*0.5
 150 GO SUB setorigin
 160 DIM X(4): DIM Y(4): DIM V(4
): DIM W(4)
 170 DATA 1,1,1,-1,-1,-1,-1,1
 179 REM Inicializa el primer cu
adrado
 180 FOR I=1 TO 4: READ X(I),Y(I
): NEXT I
 189 REM Fija el valor de MU y d
ibuja 20 cuadrados
 190 LET MU=0.1: LET UM=1-MU
 200 FOR I=1 TO 21
 208 REM Une los 4 vertices del
cuadrado (X(J),Y(J)); J=1-4
 209 REM Calcula los siguientes
4 vertices (V(J),W(J)); J=1-4
```

```
 210 LET XPT=X(4): LET YPT=Y(4):
 GO SUB moveto
 220 FOR J=1 TO 4
 230 LET XPT=X(J): LET YPT=Y(J):
 GO SUB lineto
 240 LET NJ=J+1: IF NJ=5 THEN .L
ET NJ=1
 250 LET V(J)=UM*X(J)+MU*X(NJ)
 260 LET W(J)=UM*Y(J)+MU*Y(NJ)
 270 NEXT J
 279 REM Copia las matrices V y
W en X e Y
 280 FOR J=1 TO 4
 290 LET X(J)=V(J): LET Y(J)=W(J
)
 300 NEXT J
 310 NEXT I
 320 STOP
```

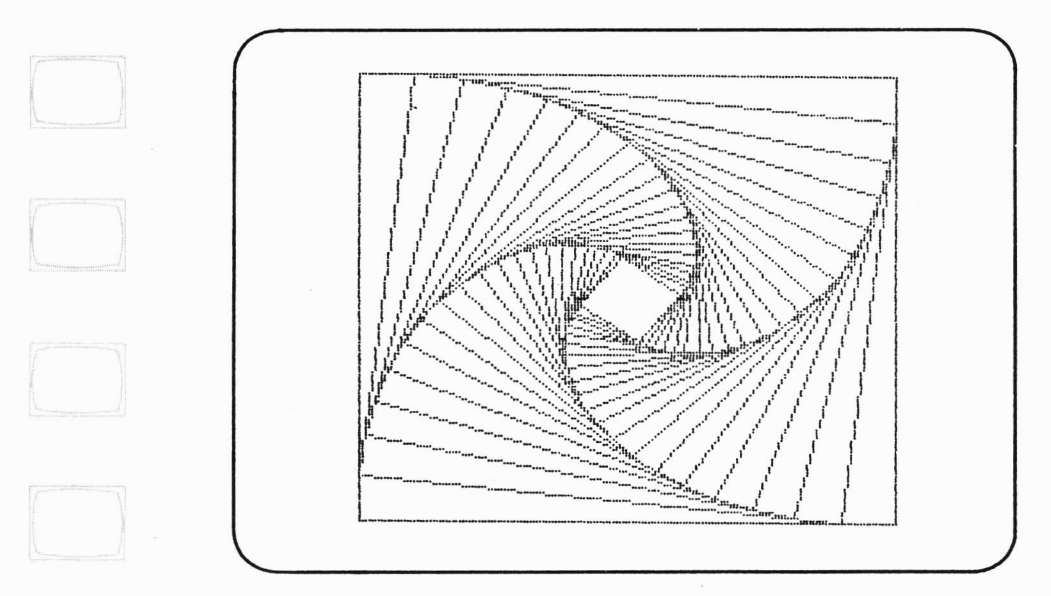


Figura 3.1

Si revolvemos un poco la ecuación de la recta en función de los vectores correspondientes a dos puntos, podemos dejarla en la forma siguiente:

$$\boldsymbol{p}_1 + \mu(\boldsymbol{p}_2 - \boldsymbol{p}_1)$$

Cuando se da en esta forma, a $\boldsymbol{p}_1$ se le denomina *vector de base*, y a $(\boldsymbol{p}_2 - \boldsymbol{p}_1)$ *vector de dirección*. De hecho cualquier punto de la recta puede actuar como base, actúa, sencillamente, como punto de fijación de una recta que es paralela al vector de dirección. El concepto de vector como identificador de una dirección merece unas ciertas consideraciones. Como hemos visto, un vector (x, y) puede representar un punto; la recta que une el origen con dicho punto puede tomarse como definición de una dirección en el plano —cualquier otra recta paralela a ella tiene, por definición, la misma dirección (y por tanto el mismo vector de dirección). Insistimos en que la recta va desde el origen al punto (x, y), éste es el llamado *sentido* positivo; una recta que va desde (x, y) hacia el origen se dice que tiene sentido negativo.

En el siguiente ejemplo se usa la interpretación dual, como punto y como dirección, de un vector.

Ejemplo 3.2

Dibuje una línea recta discontinua, con 13 tramos (y por tanto, 12 espacios entre ellos) desde el punto $\boldsymbol{p}_1 = (x_1, y_1)$ al $\boldsymbol{p}_2 = (x_2, y_2)$. Este problema se resuelve hallando 26 puntos equiespaciados en el segmento; o sea, $\boldsymbol{p}_1 + i/25\,(\boldsymbol{p}_2 - \boldsymbol{p}_1)$ donde i varía desde 0 (en $\boldsymbol{p}_1$) a 25 (en $\boldsymbol{p}_2$). De cada punto al siguiente, o bien nos movemos sin dibujar, o bien trazamos el segmento que los une. Empleamos la subrutina "plot" del CalComp (listado 2.8). No hay necesidad de guardar los valores de los puntos intermedios; tenemos $\boldsymbol{p}_1$, de forma que añadiendo $1/25\ (\boldsymbol{p}_2 - \boldsymbol{p}_1)$ al último punto calculado hallamos el siguiente hasta alcanzar $\boldsymbol{p}_2$.

Listado 3.2

```
 100 REM lineas de puntos
 110 LET start=9700: LET plot=98
00
 120 LET HORIZ=3: LET VERT=2.1
 130 GO SUB start
 140 LET XPT=HORIZ*.5: LET YPT=V
ERT*.5: LET MODE=-3
 150 GO SUB plot
 160 INPUT "TECLEA X1 E Y1 ";X1;
" , ";Y1
 170 INPUT "TECLEA X2 E Y2 ";X2;
" , ";Y2
 179 REM Se mueve al primer punt
o
 180 LET XPT=X1: LET YPT=Y1: LET
 MODE=3: GO SUB plot
 190 LET XD=(X2-X1)/25: LET YD=(
Y2-Y1)/25
```

```
 199 REM Dibuja y se mueve alter
nativamente a los siguientes 25
puntos de la recta
 200 FOR I=1 TO 25
 210 LET XPT=XPT+XD: LET YPT=YPT
+YD: LET MODE=5-MODE: GO SUB plo
t
 220 NEXT I
 230 STOP
```

Ejercicio 3.1

Experimente en el dibujo de distintos tipos de líneas a trazos: por ejemplo, *a*) los trazos el doble de largos que el espacio que los separa; *b*) fijar el tamaño de los trazos y calcular su número; *c*) los trazos podrían variar en su longitud, alternando un trazo largo y otro corto, donde la relación entre el tamaño de ambos y de uno de ellos con el espacio que los separa sean variables a introducir.

Esta representación en forma de base y dirección es de gran utilidad con vistas al cálculo del punto de intersección de dos rectas; problema éste que se presenta con bastante asiduidad en gráficos bidimensionales. Sean dos rectas $\boldsymbol{p} + \mu\boldsymbol{q}$ y $\boldsymbol{r} + \lambda\boldsymbol{s}$, donde $\boldsymbol{p} \equiv (x_1, y_1)$, $\boldsymbol{q} \equiv (x_2, y_2)$, $\boldsymbol{r} \equiv (x_3, y_3)$ y $\boldsymbol{s} \equiv (x_4, y_4)$ y $-\infty < \mu, \lambda < \infty$. Queremos encontrar los valores (únicos) de μ y λ que cumplen la condición

$$\boldsymbol{p} + \mu\boldsymbol{q} = \boldsymbol{r} + \lambda\boldsymbol{s}$$

que es, evidentemente, el punto común a ambas rectas. Esta ecuación vectorial puede desdoblarse en dos ecuaciones:

$$x_1 + \mu \times x_2 = x_3 + \lambda \times x_4 \qquad (3.1)$$

$$y_1 + \mu \times y_2 = y_3 + \lambda \times y_4 \qquad (3.2)$$

Si remodelamos estas ecuaciones obtenemos:

$$\mu \times x_2 - \lambda \times x_4 = x_3 - x_1 \qquad (3.3)$$

$$\mu \times y_2 - \lambda \times y_4 = y_3 - y_1 \qquad (3.4)$$

Multiplicando (3.3) por y_4, (3.4) por x_4 y restando ambos resultados obtenemos:

$$\mu \times (x_2 \times y_4 - y_2 \times x_4) = (x_3 - x_1) \times y_4 - (y_3 - y_1) \times x_4$$

Si $(x_2 \times y_4 - y_2 \times x_4) = 0$ entonces ambas rectas son paralelas y no existe punto de intersección (es imposible calcular μ); en caso contrario,

$$\mu = \frac{(x_3 - x_1) \times y_4 - (y_3 - y_1) \times x_4}{(x_2 \times y_4 - y_2 \times x_4)} \qquad (3.5)$$

análogamente,

$$\lambda = \frac{(x_3 - x_1) \times y_2 - (y_3 - y_1) \times x_2}{(x_2 \times y_4 - y_2 \times x_4)} \tag{3.6}$$

La solución es mucho más sencilla si una de las rectas es paralela a alguno de los ejes de coordenadas. Sea la recta $x = d$, entonces podemos tomar $\boldsymbol{r} \equiv (d, 0)$ y $\boldsymbol{s} \equiv (0, 1)$, que llevada a la ecuación (3.5) da

$$\mu = (d - x_1)/x_2$$

y si la recta es $y = d$, entonces, análogamente,

$$\mu = (d - y_1)/y_2$$

Naturalmente, si ambas rectas son paralelas, entonces el denominador de estas ecuaciones será cero y obtenemos un resultado infinito, que es donde se cortan dos rectas paralelas.

Ejemplo 3.3

Hallar el punto de intersección de las dos rectas: *a*) que pasa por los puntos $(1, -1)$ y $(-1, -3)$ y *b*) definida por los puntos $(1, 2)$ y $(2, -2)$.

Las rectas pueden escribirse

$$(1 - \mu)(1, -1) + \mu(-1, -3) \qquad -\infty < \mu < \infty \tag{3.7}$$

$$(1 - \lambda)(1, 2) + \lambda(2, -2) \qquad -\infty < \lambda < \infty \tag{3.8}$$

o bien poniéndolas en la forma vectorial base/dirección

$$(1, -1) + \mu(-2, -2) \tag{3.9}$$

$$(1, 2) + \lambda(2, -4) \tag{3.10}$$

Llevando estos valores a la ecuación (3.5) tenemos

$$\mu = \frac{(1 - 1) \times -4 - (2 + () \times 2}{(-2 \times -4 - (-2) \times 2)} = -1/2$$

Y el punto de intersección es $(1, -1) - 1/2(-2, -2) \equiv (2, 0)$.

Ejercicio 3.2

Experimente con este concepto de la representación vectorial del espacio de dos dimensiones. Usted puede generar sus propias preguntas: es fácil de comprobar si las respuestas son correctas o no. Sea el ejemplo 3.2. Sabemos que el punto $(2, 0)$ está sobre la primera recta porque usamos el valor $\mu = -1/2$ para calcularlo; la respuesta sería correcta si también estuviera sobre la segunda; lo está y $\lambda = 1/2$.

Ejercicio 3.3

Escriba un programa que lea los datos correspondientes a dos líneas rectas (dadas o bien en forma de ecuación, o bien en la forma vectorial base/dirección) y a continuación calcule (si existen) los puntos de intersección.

Recorte

A estas alturas, ya se habrá dado cuenta de que es imposible utilizar las instrucciones PLOT o DRAW con un *pixel* (x, y) que esté fuera del área gráfica, y por tanto tenemos los límites $0 \leqslant x \leqslant 225$ y $0 \leqslant y \leqslant 175$. Es muy fácil salirse inadvertidamente de esta área. De hecho, cuando se dibujan imágenes bi o tridimensionales es bastante normal construirlas de un tamaño mayor que el permitido para los gráficos en el Spectrum. Es, por tanto, necesario encontrar un algoritmo que elimine todos los tramos exteriores, reteniendo aquellas partes que deban ser dibujadas.

Supongamos que el centro de la pantalla (geométrico, no *pixel*) se corresponde con el punto $(255/2, 175/2) \equiv (127.5, 87.5)$ y por tanto, las cuatro esquinas de la zona gráfica son los puntos $(127.5 \pm 127.5, 87.5 \pm 87.5)$. Nuestro problema se reduce a calcular la parte del segmento que une el *pixel* (XA, YA) con el (XB, YB) que cae dentro de la zona (si existe). Para simplificar la solución, redefinimos el origen de coordenadas de *pixels* en el centro de la pantalla restando el vector (175.5, 87.5) de las coordenadas correspondientes al anterior sistema de coordenadas. Las esquinas del rectángulo gráfico tienen como coordenadas $(\pm 127.5, \pm 87.5)$. Extendemos los lados del rectángulo dividiendo, por tanto, el espacio en nueve sectores; vea la figura 3.2, que muestra no sólo el área gráfica, sino también el BORDER. En este diagrama, se han dibujado varios segmentos como ayuda a la explicación del algoritmo. Cualquier punto del espacio puede clasificarse por dos parámetros IX e IY donde

1) IX $= -1, 0$ ó $+1$ dependiendo de si la coordenada x del punto está a la izquierda, dentro, o a la derecha de la zona gráfica;

2) IY $= -1, 0$ ó $+1$ dependiendo de si la coordenada y del punto está abajo, dentro, o sobre el rectángulo gráfico.

Estos valores se calculan, cuando sea necesario, dentro del programa del algoritmo.

Si los dos puntos extremos del segmento —o sea (XA, YA) y (XB, YB)— tienen como parámetros IXA e IYA, IXB e IYB respectivamente, entonces se nos presentan varias posibilidades:

i) IXA = IXB $\neq 0$ ó IYA = IYB $\neq 0$. El segmento completo está fuera del rectángulo y puede ser ignorado sin problemas. Por ejemplo la línea AB de la figura 3.2.

ii) IXA = IXB = IYA = IYB = 0. El segmento está totalmente dentro del rectángulo y debe dibujarse. Por ejemplo la línea CD.

iii) Los casos restantes deben considerarse más detenidamente. Si IXA $\neq$ 0 y/o IYA $\neq$ 0, entonces el punto (XA, YA) está fuera del rectángulo y deben hallarse otros nuevos valores para XA e YA; para evitar confusiones los denominaremos XA′ e YA′. (XA′, YA′) es el punto del segmento más próximo a (XA, YA) donde la línea corta a la zona gráfica. La fórmula para hallar este punto la vimos antes; es la intersección del segmento con una recta paralela a un eje de coordenadas. Si el segmento cae fuera del rectángulo, entonces definimos el punto (XA′, YA′) como el de intersección del segmento con la prolongación de los bordes verticales. Si IXA = IYA = 0 entonces (XA′, YA′) $\equiv$ (XA, YA). El punto (XB′, YB′) se calcula de una forma similar; vea el algoritmo dado por la subrutina "recorte" en el listado 3.3 La parte del segmento a dibujar es la que une los puntos (XA′, YA′) con el (XB′, YB′). Si el segmento está totalmente fuera del rectángulo, el algoritmo nos da (XA′, YA′) = (XB′, YB′) y el segmento degenera en un punto y es ignorado. Por ejemplo, EF; tras ser recortado, queda E′F′; GH queda GH′ (G = G′) e IJ degenera en el punto I′ = J′.

En resumen, la subrutina "recorte" toma los *pixels* extremos del segmento (XA, YA) y (XB, YB), y los transforma con respecto al sistema centrado en el rectángulo gráfico. A continuación determina cuál de las tres posibilidades anteriores se cumple y actúa en consecuencia: i) salir inmediatamente de la subrutina; ii) unir ambos puntos; o iii) calcular los puntos "acentuados" y unirlo con un segmento.

El listado 3.3 incluye también una nueva versión de la subrutina "trazalíneas" que llama a "recorte" en vez de usar las instrucciones PLOT y DRAW, permitiendo el trazado de segmentos rectilíneos independientemente de su posición en el espacio. De ahora en adelante usaremos esta nueva versión de "trazalíneas". Demostrará que vale sus bytes en oro, especialmente en el estudio de objetos tridimensionales.

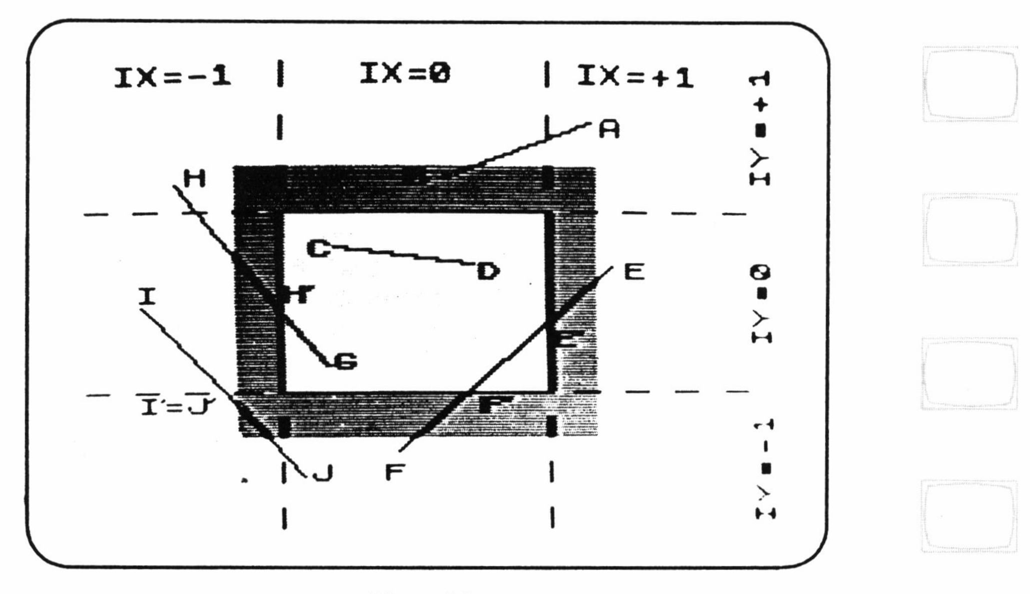


Figura 3.2

Ejercicio 3.4

Emplee esta versión modificada en los programas del capítulo 2. Elija los valores de HORIZ y VERT de forma que algunas líneas se salgan de la zona gráfica.

Listado 3.3

```
8400 REM recorte
8401 REM Datos de entrada XA,YA,
XB,YB
8409 REM Cambia el sistema de co
ordenadas
8410 LET XA=XA-127.5: LET YA=YA-
87.5: LET XB=XB-127.5: LET YB=YB
-87.5
8419 REM Encuentra el valor del
sector de los puntos (XA,YA),(XB
,YB)
8420 LET IXA=0: IF ABS XA>127.5
THEN  LET IXA=SGN XA
8430 LET IYA=0: IF ABS YA>87.5 T
HEN  LET IYA=SGN YA
8440 LET IXB=0: IF ABS XB>127.5
THEN  LET IXB=SGN XB
8450 LET IYB=0: IF ABS YB>87.5 T
HEN  LET IYB=SGN YB
8459 REM Si los puntos estan en
el mismo sector fuera de pantall
a volver
8460 IF IXA*IXB=1 OR IYA*IYB=1 T
HEN  RETURN
8470 IF IXA=0 THEN  GO TO 8500
8479 REM Mueve el primer punto a
l borde X mas cercano
8480 LET XX=127.5*IXA: LET YA=YA
+(YB-YA)*(XX-XA)/(XB-XA): LET XA
=XX
8490 LET IYA=0: IF ABS YA>87.5 T
HEN  LET IYA=SGN YA
8500 IF IYA=0 THEN  GO TO 8520
8509 REM Mueve el primer punto a
l borde Y mas cercano
8510 LET YY=87.5*IYA: LET XA=XA+
```

```
(XB-XA)*(YY-YA)/(YB-YA): LET YA=
YY
8520 IF IXB=0 THEN  GO TO 8550
8529 REM Mueve el segundo punto
al borde X mas cercano
8530 LET XX=127.5*IXB: LET YB=YA
+(YB-YA)*(XX-XA)/(XB-XA): LET XB
=XX
8540 LET IYB=0: IF ABS YB>87.5 T
HEN  LET IYB=SGN YB
8550 IF IYB=0 THEN  GO TO 8570
8559 REM Mueve el segundo punto
al borde Y mas cercano
8560 LET YY=87.5*IYB: LET XB=XA+
(XB-XA)*(YY-YA)/(YB-YA): LET YB=
YY
8570 IF ABS (XA-XB)<0.000001 AND
 ABS (YA-YB)<0.000001 THEN  RETU
RN
8579 REM Traza los puntos no coi
ncidentes
8580 LET XA=INT (XA+128): LET YA
=INT (YA+88): LET XB=INT (XB+128
):                         LET YB=INT
(YB+88)
8590 PLOT XA,YA: DRAW XB-XA,YB-Y
A: RETURN

9400  REM linea/recorte
9401 REM Datos de entrada XPT,YP
T,XPEN,YPEN
9409 REM Datos de salida XPEN,YP
EN
9410 LET XA=XPEN: LET YA=YPEN
9420 LET XPEN=FN X(XPT)
9430 LET YPEN=FN Y(YPT)
9440 LET XB=XPEN: LET YB=YPEN
9450 GO SUB clip
9460 RETURN
```

Volviendo al empleo de un vector ($\boldsymbol{q} \equiv (x, y) \neq (0, 0)$) para representar una dirección, debemos darnos cuenta de que el vector producto de $\boldsymbol{q}$ por un escalar $k > 0$, representa *la misma dirección y sentido* que $\boldsymbol{q}$. (Si $k < 0$ entonces tiene la misma

dirección pero el sentido es el opuesto). En particular, si $k = 1/|\boldsymbol{q}|$, entonces $k\boldsymbol{q} = (x/\sqrt{(x^2 + y^2)}, y/\sqrt{(x^2 + y^2)})$ y su valor absoluto es 1.

Por tanto, un punto cualquiera de la recta $\boldsymbol{p} + \mu\boldsymbol{q}$, está a una distancia $|\mu\boldsymbol{q}|$ del punto base $\boldsymbol{p}$, y si $|\boldsymbol{q}| = 1$ (un vector unitario), entonces la distancia del punto a $\boldsymbol{p}$ es $|\mu|$.

Consideraremos ahora los ángulos formados por los vectores de dirección con varias direcciones fijas. Sea α el ángulo formado por la recta que une el origen $\boldsymbol{O}$ con el punto $\boldsymbol{q} \equiv (x, y)$, y la parte positiva del eje x. Entonces $x = |\boldsymbol{q}| \times \cos\alpha$ e $y = |\boldsymbol{q}| \times \text{sen}\ \alpha$; vea la figura 3.3; se pueden construir figuras similares para los otros tres cuadrantes.

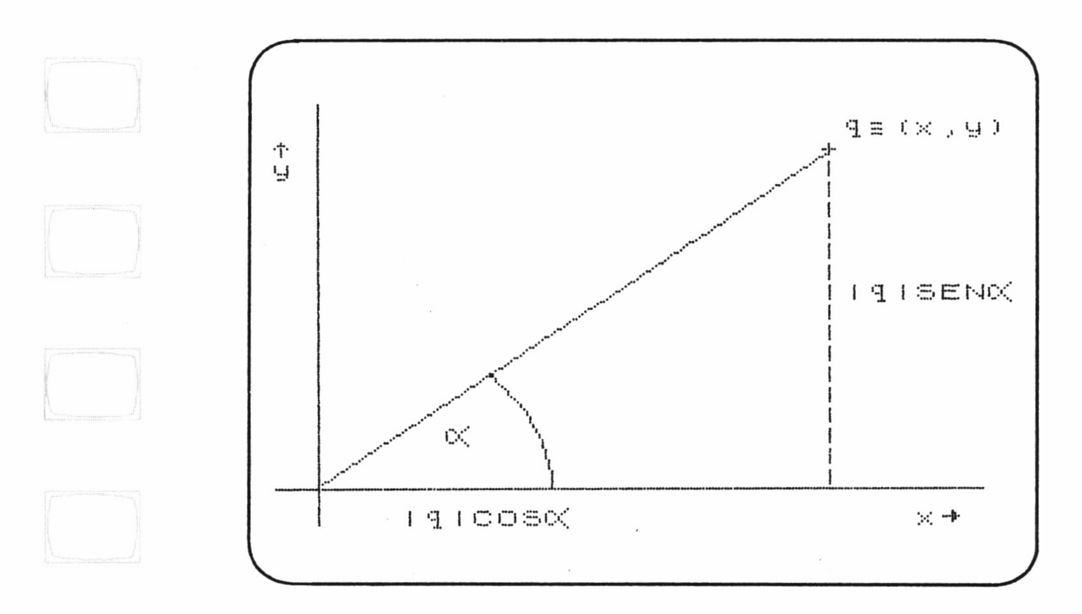


Figura 3.3

Si $\boldsymbol{q}$ es un vector unitario ($|\boldsymbol{q}| = 1$), entonces $\boldsymbol{q} \equiv (\cos\ \alpha,\ \text{sen}\ \alpha)$. Si tenemos en cuenta que $\text{sen}\,\alpha = \cos(\alpha - \pi/2)$ para cualquier α, podemos escribir $\boldsymbol{q} = (\cos\alpha, \cos(\alpha - \pi/2))$, pero $\alpha - \pi/2$ es el ángulo que forma el vector con la parte positiva del eje y. A causa de este hecho se denominan *cosenos directores* a las coordenadas de un vector de dirección unitario, ya que son los cosenos de los ángulos que forma el vector con los semiejes positivos respectivos.

Antes de continuar, deberíamos echar un vistazo a las funciones trigonométricas de que dispone el BASIC: SIN (seno) y COS (coseno), y a la función inversa ATN (arco tangente). SIN y COS son funciones con un solo parámetro (un ángulo dado en radianes) y un resultado (un valor entre -1 y $+1$). La función ATN acepta cualquier valor y calcula el ángulo en radianes (en el llamado *rango principal* entre $-\pi/2$ y $+\pi/2$) cuya tangente es ese valor.

Esto nos conduce al problema de encontrar el ángulo que forma una dirección

cualquiera $\boldsymbol{q} \equiv (x, y)$ con el semieje positivo x. El problema lo resuelve la subrutina "ángulo" dada en el listado 3.4; "ángulo" será de gran utilidad en los capítulos siguientes cuando consideremos el espacio de tres dimensiones.

Listado 3.4

```
8800 REM angulo
8801 REM Datos de entrada AX,AY
8802 REM Datos de salida TETA
8809 REM TETA es el angulo entre
 la recta a (AX,AY) y el eje X p
ositivo
8810 IF ABS AX>0.00001 THEN  GO
TO 8860
8819 REM La recta es vertical
8820 LET THETA=PI/2
8830 IF AY<0 THEN  LET THETA=THE
TA+PI
8840 IF ABS AY<0.00001 THEN  LET
 THETA=0
8850 RETURN
8859 REM La recta no es vertical
, tiene tangente finita
8860 LET THETA=ATN (AY/AX)
8870 IF AX<0 THEN  LET THETA=THE
TA+PI
8880 RETURN
```

Supongamos ahora que tenemos dos vectores de dirección (a, b) y (c, d); por simplicidad los consideraremos unitarios y colocados sobre el origen de coordenadas (véase la figura 3.4). Queremos calcular el ángulo agudo α que forman estas rectas. A partir de la figura podemos establecer que $\mathrm{OA} = \sqrt{(a^2 + b^2)} = 1$ y $\mathrm{OB} = \sqrt{(c^2 + d^2)} = 1$. De forma que por la regla del coseno

$$\mathrm{AB}^2 = \mathrm{OA}^2 + \mathrm{OB}^2 - 2\mathrm{OA} \times \mathrm{OB} \times \cos\alpha = 2 \times (1 - \cos\alpha)$$

Pero también por el teorema de Pitágoras:

$$\begin{aligned}\mathrm{AB}^2 &= (a - c)^2 + (b - d)^2 = (a^2 + b^2) + (c^2 + d^2) - 2(a \times c + b \times d)\\ &= 2 - 2(a \times c + b \times d)\end{aligned}$$

Por tanto $a \times c + b \times d = \cos\alpha$. Es posible que $a \times c + b \times d$ sea negativo, en cuyo caso $\cos^{-1}(a \times c + b \times d)$ es obtuso y el ángulo agudo que buscamos sería

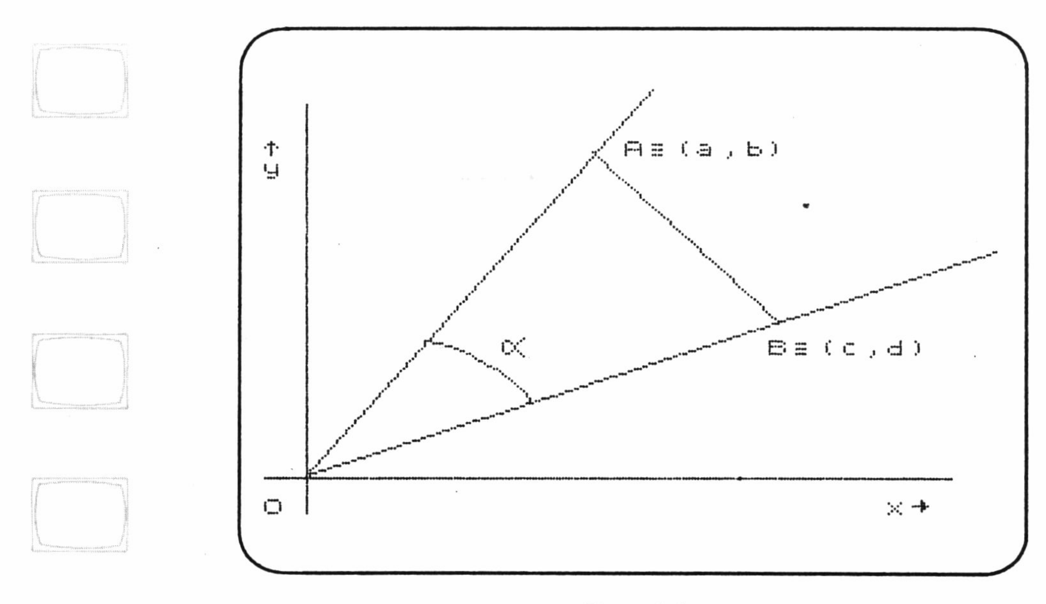


Figura 3.4

$\pi - \alpha$. Como $\cos(\pi - \alpha) = -\cos\alpha$, entonces el ángulo agudo se puede calcular como $\cos^{-1}(|a \times c + b \times d|)$. Por ejemplo, dadas dos rectas cuyos cosenos directores son $(\sqrt{(3/2}, 1/2)$ y $(-1/2, -\sqrt{3/2}))$, tenemos que $a \times b + b \times d = -\sqrt{(3/2)}$ y por tanto $\alpha = \cos^{-1}(\sqrt{(3/2)}) = \pi/6$. Este ejemplo sencillo nos presenta el concepto de *producto escalar* · de dos vectores, $(a, b) \cdot (c, d) = a \times c + b \times d$. El producto escalar se puede extender a espacios con mayor número de dimensiones (véase el capítulo 7 donde se presenta un ejemplo entre dimensiones) y siempre tiene la propiedad de dar el coseno del ángulo entre cualquier par de rectas cuyas direcciones están definidas por los dos vectores.

Curvas: Representación en forma de función y en formas paramétricas

Una curva en un espacio bidimensional puede considerarse como una relación entre los valores de las coordenadas x e y de sus puntos, lo que se llama una *relación funcional.* Por otra parte, ambas coordenadas pueden darse en términos de otras variables o parámetros; esta forma de definir una curva se conoce como *forma paramétrica.*

Ya hemos visto que una recta (un arco de circunferencia de radio infinito) puede expresarse como $ay = bx + c$. Si retocamos la ecuación de forma que su término derecho sea cero ($ay - bx - c = 0$), entonces la expresión igualada a cero es lo que llamaremos una representación funcional de la recta y lo escribiremos:

$$f(x, y) \equiv ay - bx - c$$

Todos los puntos, y solamente ellos, que cumplan la propiedad $f(x, y) = 0$ estarán sobre la línea. Esta representación divide a todos los puntos del espacio bidimen-

sional en tres grupos, $f(x, y) = 0$ (el conjunto cero), $f(x, y) > 0$ (el positivo) y $f(x, y) < 0$ (el negativo). Si la función divide al espacio solamente en la curva y dos *áreas conexas* (decimos que un área es conexa si dos puntos cualesquiera de ella pueden unirse por una curva que no corta a la curva que limita al área), entonces podemos identificar a estas áreas con los conjuntos positivo y negativo de f. Sin embargo, ¡cuidado!, existen muchas funciones elementales (por ejemplo, $g(x, y) = \cos(y) - \operatorname{sen}(x)$) que definen no una sino una serie de curvas y por tanto, dividen el espacio incluso en un número infinito de áreas conexas (obsérvese que $g(x, y) = g(x + 2m\pi, y + 2n\pi)$ para cualquier m y n enteros). De forma que es posible que dos áreas inconexas pertenezcan al conjunto positivo.

Obsérvese también que la representación no es única. Podíamos haber definido la recta de otra forma equivalente,

$$f'(x, y) \equiv bx + c - ay$$

en cuyo caso el conjunto positivo de esta función es el negativo de la $f(x, y)$, y viceversa.

Cuando la curva divide el espacio en dos áreas conexas es muy útil en el trazado gráfico con ordenador, como veremos en el estudio de algoritmos gráficos en dos y (especialmente) tres dimensiones. Por ejemplo, tomemos la línea recta

$$f(x, y) \equiv ay - bx - c$$

donde un punto (x_1, y_1) está en el mismo semiplano que (x_2, y_2) si y sólo si $f(x_1, y_1)$ tiene el mismo signo que $f(x_2, y_2)$. La representación funcional nos da más información sobre el punto que la de saber si está o no sobre la línea: también nos permite calcular su distancia a la recta.

Sea la recta anterior, cuyo vector de dirección es el (a, b). Una recta perpendicular a ella tendrá un vector de dirección $(-b, a)$ (recuérdese que $\cos(\alpha - \pi/2) = \operatorname{sen}\alpha$ y $\operatorname{sen}(\alpha - \pi/2) = -\cos\alpha$). De forma que el punto $\boldsymbol{q}$, sobre la recta, más próximo al $\boldsymbol{p} \equiv (x_1, y_1)$ es de la forma

$$\boldsymbol{q} \equiv (x_1, y_1) + \mu(-b, a)$$

que es una nueva recta que une $\boldsymbol{p}$ y $\boldsymbol{q}$ y que además es perpendicular a la recta anterior. Como $\boldsymbol{q}$ está también sobre dicha recta

$$f(\boldsymbol{q}) = f((x_1, y_1) + \mu(-b, a)) = 0$$

por tanto

$$a \times (y_1 + \mu \times a) - b \times (x_1 - \mu \times b) - c = f(x_1, y_1) + \mu(a^2 + b^2) = 0$$

y obtenemos $\mu = -f(x_1, y_1)/(a^2 + b^2)$. El punto $\boldsymbol{q}$ está a una distancia $\mu \times |(-b, a)|$ del (x_1, y_1), lo que significa, naturalmente, que la distancia del punto (x_1, y_1) a la recta es $\mu \times \sqrt{(a^2 + b^2)} = -f(x_1, y_1)/\sqrt{(a^2 + b^2)}$; si el signo nos indica el semiplano sobre el que se encuentra el punto. Si $a^2 + b^2 = 1$, entonces $|f(x_1, y_1)|$ nos da la distancia del punto (x_1, y_1) a la recta.

Esta idea nos conduce directamente a un sistema para implementar *superficies convexas;* esto es, una zona tal que cualquier par de puntos dentro de ella pueden

unirse por un segmento incluido totalmente en dicha área. Nos limitaremos al estudio de polígonos convexos; cualquier curva convexa puede aproximarse por un polígono convexo de gran número de lados.

Suponga que tenemos un polígono convexo con n vértices $\{\boldsymbol{p}_i \equiv (x_i, y_i) | i = 1, ..., n\}$ ordenados en sentido horario o antihorario (coincidente o contrario a las agujas del reloj, respectivamente); denominaremos a esta forma de describir un polígono convexo como *conjunto orientado de vértices*. El problema de hallar el sentido de giro lo trataremos en el capítulo 7. Los n segmentos que limitan al polígono vendrán dados por las ecuaciones

$$f_i(x, y) \equiv (x_{i+1} - x_i) \times (y - y_i) - (y_{i+1} - y_i) \times (x - x_i)$$

donde $i = 1, ..., n$, y la suma de los subíndices se realiza módulo n (esto es, $n + j \equiv j$ para $1 \leqslant j < n$). ¡Trate de demostrar que la ecuación anterior es realmente la de los lados del polígono!

La definición sistemática de los lados del polígono nos permite definir su parte interna. Dado cualquier lado, por ejemplo el que une los vértices $\boldsymbol{p}_i$ y $\boldsymbol{p}_{i+1}$, se cumple que todos los puntos interiores del polígono están situados en la misma banda que los restantes vértices del polígono, en particular que $\boldsymbol{p}_{i+2}$. Por tanto la parte interna está dada por:

$$\{(x, y) | \text{signo de } f_i(x, y) = \text{signo de } f_i(x_{i+2}, y_{i+2}) \neq 0 : i = 1, ..., n\}$$

Un punto del perímetro es:

$\{(x, y) |$existe un j, o dos si (x, y) es una esquina, donde $1 \leqslant j \leqslant n$ tal que $f_j(x, y) = 0$ y signo de $f_i(x, y) =$ signo de $f_i(x_{i+2}, y_{i+2}) \neq 0 : \neq j$ y $1 \leq i \leq n\}$

Un punto exterior al área está definido por

$$\{(x, y) | \text{existe un } j, 1 \leqslant j \leqslant n \text{ tal que } 0 \neq \text{signo de } f_j(x, y) \neq \text{signo de } f_j(x_{j+2}, y_{j+2}) \neq 0\}$$

Naturalmente todas las sumas de índices se hacen módulo n.

Ejemplo 3.4

Suponga que nos dan un polígono convexo cuyos vértices son (1, 0), (5, 2), (4, 4) y (−2, 1) (vea la figura 3.5). La orientación es antihoraria. Los puntos (3, 2), (1, 4) y (3, 1), ¿están dentro, fuera o sobre algún lado del polígono? ¿Cuál es la distancia del punto (4, 4) a la primera línea?

$$f_1(x, y) \equiv (5 - 1) \times (y - 0) - (2 - 0) \times (x - 1) \equiv 4y - 2x + 2$$
$$f_2(x, y) \equiv (4 - 5) \times (y - 2) - (4 - 2) \times (x - 5) \equiv -y - 2x + 12$$
$$f_3(x, y) \equiv (-2 - 4) \times (y - 4) - (1 - 4) \times (x - 4) \equiv -6y + 3x + 12$$
$$f_4(x, y) \equiv (1 + 2) \times (y - 1) - (0 - 1) \times (x + 2) \equiv 3y + x - 1$$

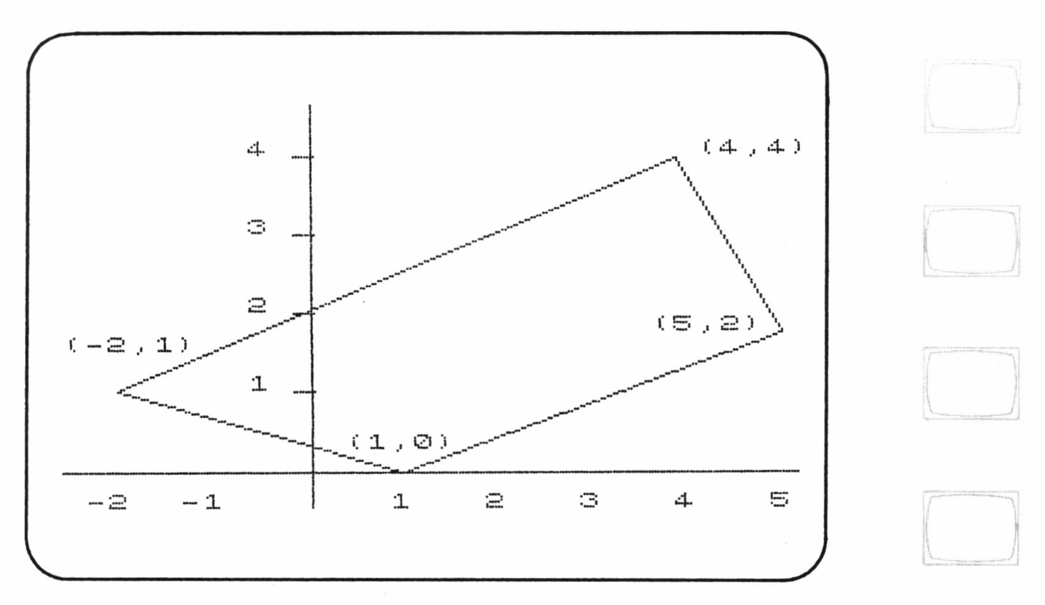


Figura 3.5

Por tanto el punto (3, 2) es interno ya que $f_1(3, 2) = 4$ y $f_1(4, 4) = 10$; $f_2(3, 2) = 4$ y $f_2(-2, 1) = 15$; $f_3(3, 2) = 9$ y $f_3(1, 0) = 15$; $f_4(3, 2) = 8$ y $f_4(5, 2) = 10$; todos tienen signo positivo.

El punto (1, 4) está fuera ya que $f_3(1, 4) = -9$ y $f_3(1, 0) = 15$: signos contrarios.

El punto (3, 1) pertenece al perímetro ya que $f_1(3, 1) = 0$, $f_2(3, 1) = 5$, $f_3(3, 1) = 15$ y $f_4(3, 1) = 5$.

De hecho, no hay necesidad de calcular $f_i(x_{i+2}, y_{i+2})$ para todo i, todos tienen el mismo signo; de forma que una vez calculado $f_1(x_3, y_3)$ entonces podemos seguir con este mismo valor.

La distancia del punto (4, 4) a la primera recta $f_1(4, 4)/\sqrt{(4^2 + 2^2)} = 10/\sqrt{20} =$ $= \sqrt{5}$.

Ejercicio 3.5

Imagine dos polígonos convexos que se intersectan mutuamente. El área común a ambos es también un polígono convexo. Emplee los métodos mencionados en este capítulo para calcular los vértices del nuevo polígono.

Habiendo trabajo con la representación funcional de una recta, ¿qué tal si empleamos la forma paramétrica? Indicamos antes que esta forma consiste en dar las coordenadas x e y de un punto cualquiera de la curva en función de un(os) parámetro(s) (que pueden ser las propias coordenadas x e y), junto con su rango de variación. Pero ya hemos visto la forma paramétrica de una recta: es simplemente la representación como base y dirección.

$$\begin{aligned} \boldsymbol{b} + \mu \boldsymbol{d} &\equiv (x_1, y_1) + \mu(x_2, y_2) \\ &\equiv (x_1 + \mu \times x_2, y_1 + \mu \times y_2) \quad \text{con} \quad -\infty < \mu < \infty \end{aligned}$$

μ es el parámetro, y $x_1 + \mu \times x_2$ e $y_1 + \mu \times y_2$ son las coordenadas x e y respectivamente de un punto de la recta (x, y) que dependen sólo de μ.

Podemos construir también representaciones funcionales y formas paramétricas para la mayoría de las curvas continuas. Por ejemplo, una sinusoide está dada por $f(x, y) \equiv y - \text{sen}(x)$ (representación funcional), y por $(x, \text{sen}(x))$ con $-\infty < x < \infty$ (forma paramétrica). Una sección cónica general (elipse, parábola o hipérbola) viene dada por la función

$$f(x, y) \equiv a \times x^2 + b \times y^2 + h \times x \times y + f \times x + g \times y + c$$

donde los coeficientes a, b, c, f, g, h definen una única curva. Una circunferencia centrada en el origen y de radio r tiene $a = b = 1, f = g = h = 0$ y $c = -r^2$, y nos queda $f(x, y) \equiv x^2 + y^2 - r^2$. Todos los puntos (x, y) de la circunferencia cumplen $f(x, y) = 0$, los interiores del círculo son aquellos tales que $f(x, y) < 0$; correspondientemente, los exteriores hacen $f(x, y) > 0$. La forma paramétrica de la circunferencia es $(r \cos \alpha, r \,\text{sen}\, \alpha)$ donde $0 \leqslant \alpha \leqslant 2\pi$. (Ya vimos anteriormente las formas paramétricas de la circunferencia, elipse y espiral en el capítulo 2.)

Es muy útil experimentar con estos (y otros) conceptos en geometría bidimensional. Se presentarán muchas ocasiones donde sea necesario incluir estas ideas en algún programa, además de la necesidad siempre presente de generar datos para la construcción de diagramas.

Ejemplo 3.5

Suponga que deseamos dibujar una bola esférica (radio r) que desaparece dentro de un agujero elíptico (eje mayor a, eje menor b), véase la figura 3.6. Tanto la elipse como la esfera tienen zonas tapadas.

Coloquemos la elipse centrada en el origen con el eje mayor horizontal, y el centro de la circunferencia a una distancia d sobre el origen. La representación funcional de la elipse es

$$f_e(x, y) \equiv x^2/a^2 + y^2/b^2 - 1$$

y en forma paramétrica

$$(a \times \cos \alpha, b \times \sin \alpha) \qquad \text{con } 0 \leqslant \alpha \leqslant 2\pi$$

Para la circunferencia

$$f_c(x, y) \equiv x^2 + (y - d)^2 - r^2$$

y en forma paramétrica

$$(r \times \cos \lambda, d + r \times \sin \lambda) \quad \text{donde} \quad 0 \leqslant \lambda \leqslant 2\pi$$

Para generar el dibujo debemos encontrar los puntos (x, y) comunes a la circunferencia y a la elipse (si los hay). Como demostración útil emplearemos ambas representaciones en la búsqueda de la solución (representación funcional para el círculo y paramétrica para la elipse).

Así que buscamos los puntos $(x, y) \equiv (a \times \cos \alpha, b \times \text{sen}\ \alpha)$ de la elipse, que también satisfacen $f_c(x, y) = 0$. O sea

$$a^2 \times \cos^2 \alpha + (b \times \sin \alpha - d)^2 - r^2 = 0$$

y $$a^2 \times \cos^2 \alpha + b^2 \times \sin^2 \alpha - 2 \times b \times d \times \sin \alpha + d^2 - r^2 = 0$$

Y como $\cos^2 \alpha = 1 - \text{sen}^2 \alpha$

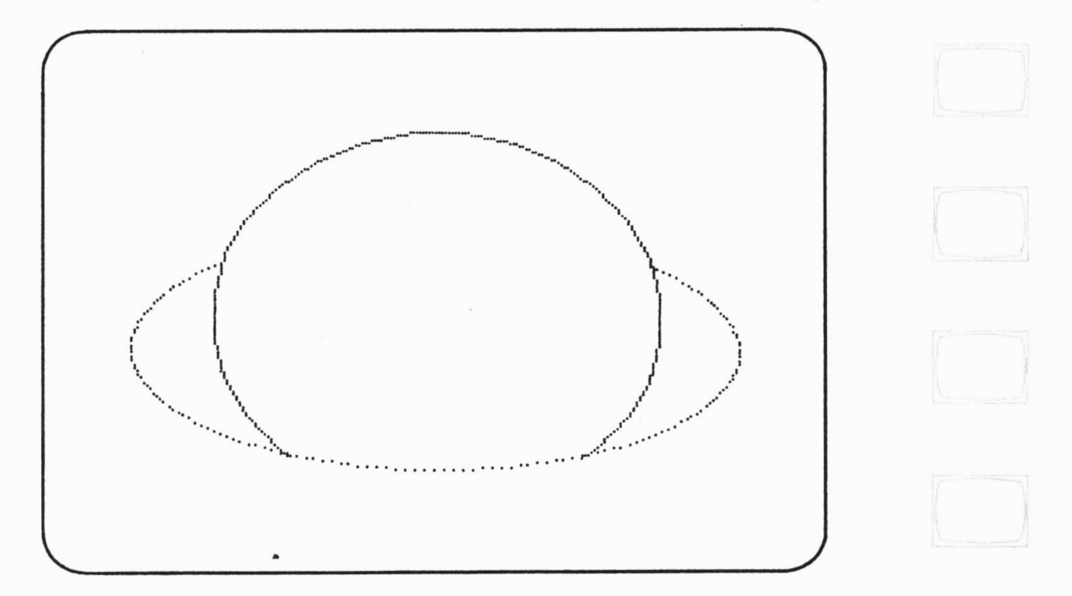
$$(b^2 - a^2) \times \sin^2 \alpha - 2 \times b \times d \times \sin \alpha + a^2 + d^2 - r^2 = 0$$

Esta ecuación es una cuadrática cuya incógnita es sen α, que se puede resolver fácilmente (la ecuación cuadrática $Ax^2 + Bx + C = 0$ tiene dos raíces $(-B \pm \sqrt{(B^2 - 4 \times A \times C)})/(2 \times A)$). Para cada valor de sen α podemos encontrar valores para α dentro del rango $0 \leqslant \alpha \leqslant 2\pi$ (si existe) y los puntos de intersección $(a \times \cos \alpha, b \times \text{sen}\ \alpha)$.

No existe una regla exacta y rápida que nos indique cuál representación es la más idónea para una situación dada: se precisa un sexto sentido que sólo se logra con la experiencia.

Ejercicio 3.6

Escriba un programa que dibuje la figura 3.6.

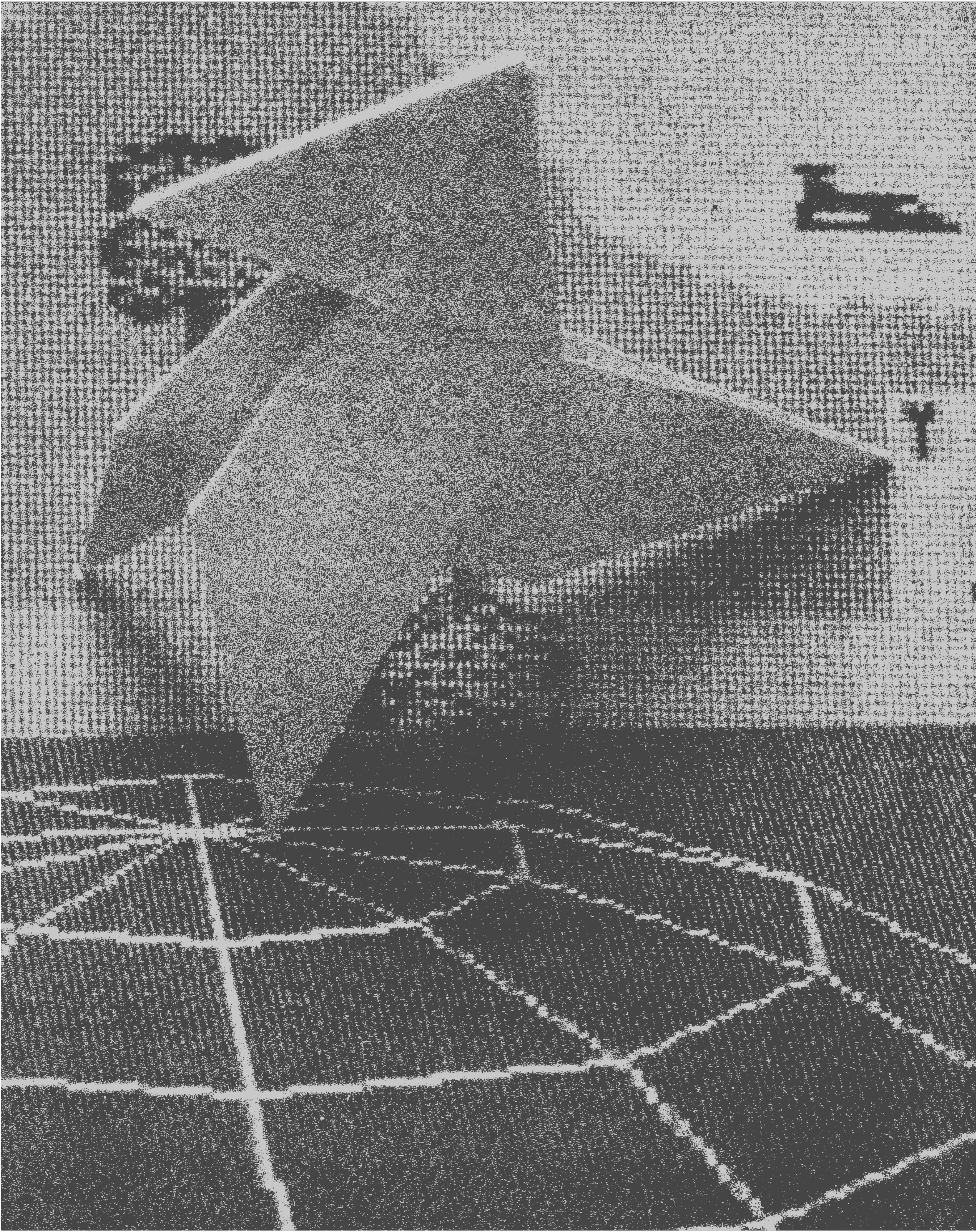


Programas completos

I. "rut1" y el listado 3.1: no precisa datos de entrada.

II. Listados 2.1, 2.8 y 3.2: los datos de entrada son los dos pares de coordenadas (X1, Y1) y (X2, Y2), donde $-3 < X1, X2 < 3$ y $-2.1 < Y1, Y2 < 2.1$.
Nota: desde este punto, el listado 3.3 "recorte" *(clip)* y una nueva versión de "trazalíneas" *(lineto)* sustituirá el listado 2.5 en "rut1".

III. Lo mismo que en I, pero con la nueva "rut1": cambie HORIZ a 1.5 y VERT a 1.



4

Representación matricial de las transformaciones en el espacio bidimensional

En el capítulo 2 contemplamos la necesidad de mover por la pantalla los dibujos de los objetos. En vez de cambiar continuamente el sistema de coordenadas de la pantalla, es mucho más fácil desde el punto de vista conceptual definir el objeto en su forma más sencilla (dando sus vértices como *pixels* o como valores de sus coordenadas, junto con la información adicional necesaria para el trazado de las líneas y superficies correspondientes a los vértices), y a continuación, trasladar dicho objeto a la pantalla en la posición que nos interese dejando el sistema de coordenadas fijo. De todas las transformaciones posibles, restringiremos nuestra actuación a las del tipo lineal (si no conoce el concepto de transformaciones lineales, espérese unas líneas y lo definiremos). A menudo tendremos que transformar un conjunto grande de vértices y la manera más eficiente de hacer los cálculos es mediante el empleo de *matrices.*

Antes de tratar la representación matricial de las transformaciones, deberíamos explicar lo que entendemos por matriz, y también por *vector columna.* En particular, trabajaremos con matrices cuadradas; 3 × 3 para el estudio en dos dimensiones, y de 4 × 4 cuando consideramos el espacio tridimensional. Una matriz 3 × 3 (por ejemplo A) es simplemente un conjunto de números reales que, por comodidad, agrupamos en un bloque de 3 filas y 3 columnas; un vector columna (por ejemplo D) es un conjunto de números colocados en una columna con 3 filas.

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \quad \text{y} \quad \begin{pmatrix} D_1 \\ D_2 \\ D_3 \end{pmatrix}$$

Un elemento cualquiera de la matriz A se representa comúnmente por A_{ij}, el primer subíndice (i) nos dice que el elemento se encuentra sobre la i-ésima fila, y el segundo (j) que pertenece a la j-ésima columna (por ejemplo, el elemento A_{23} representa al número colocado en la segunda fila y en la tercera columna). De una forma similar D_i representa el elemento de la fila i-ésima de un vector columna. Cuando trabajemos con matrices y vectores columna, estos subíndices serán calculados y obtendremos sus valores numéricos mediante expresiones enteras. Es importante señalar que la *información* que nos proporciona una matriz o vector columna, está tanto en los valores de sus elementos, como en el orden y posición en que están colocados. En un programa BASIC (como en cualquier otro lenguaje de ordenadores), las sentencias se escriben en una línea sin permitirse el uso de subíndices ni superíndices, y por tanto las matrices y vectores se implementan como *arrays** y los valores de los subíndices aparecen entre paréntesis detrás del identificador del *array* (así, A_{ij} se representará como $A(i, j)$).

Existe la suma de matrices. La matriz $C = A + B$ (C es la matriz suma de las matrices A y B) se define por su término general C_{ij} como

$$C_{ij} = A_{ij} + B_{ij} \quad 1 \leqslant i,j \leqslant 3$$

También podemos multiplicar una matriz A por un escalar k y el resultado será otra matriz B definida por

$$B_{ij} = k \times A_{ij} \quad 1 \leqslant i,j \leqslant 3$$

Podemos, asimismo, multiplicar una matriz A por un vector columna D y obtenemos como resultado otro vector columna E, como

$$E_i = A_{i1} \times D_1 + A_{i2} \times D_2 + A_{i3} \times D_3 = \sum_k A_{ik} \times D_k \quad \text{donde} \quad 1 \leqslant i \leqslant 3$$

El elemento de la i-ésima fila del nuevo vector columna es la suma de los productos de los elementos correspondientes de la fila i-ésima de la matriz con los del vector columna.

Más aún, podemos calcular el producto (matricial) $C = A \times B$ de dos matrices A y B.

$$C_{ij} = A_{i1} \times B_{1j} + A_{i2} \times B_{2j} + A_{i3} \times B_{3j} = \sum_k A_{ik} \times B_{kj} \quad \text{donde} \quad 1 \leqslant i,j \leqslant 3$$

Tomamos la suma (ordenada) de los elementos de la j-ésima fila de la primera matriz multiplicados por los elementos de la j-ésima columna de la segunda. Para

* *N. del T:* Hemos preferido en este capítulo mantener la palabra inglesa "array" debido a su uso común en la jerga informática, en lugar de su traducción como "arreglo", "vector" o "matriz" que podrían inducir a error.

quien no esté familiarizado con el cálculo matricial, puede resultar extraño que $A \times B \neq B \times A$, en general (¡el producto de matrices no es *conmutativo*!). Por ejemplo

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ pero } \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Experimente con estas ideas hasta que tenga la suficiente soltura para emplearlas en la teoría que sigue a continuación. Para aquellos que deseen más detalles sobre la teoría de matrices, podrán encontrarla en cualquier libro de álgebra elemental.

Existe una matriz especial conocida como la *matriz identidad* I (también denominada matriz unidad).

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

También podemos calcular el *determinante* de una matriz: $\det(A)$

$$\det(A) = A_{11}(A_{22} \times A_{33} - A_{23} \times A_{32}) + A_{12} \times (A_{23} \times A_{31} - A_{21} \times A_{33}) + A_{13} \times (A_{21} \times A_{32} - A_{22} \times A_{31})$$

Si el determinante de una matriz es cero, entonces se dice que la matriz es *singular*, siendo *no singular* en caso contrario. Todas las matrices no singulares tienen su matriz *inversa* A^{-1}, que cumple las condiciones $A \times A^{-1} = A^{-1} \times A = I$. Existen varios métodos para calcular la inversa de una matriz: en el capítulo 7 (listado 7.5) damos un programa que lo realiza por el método de los Adjuntos.

Apliquemos lo anterior a la transformación de puntos en el espacio. Sea un punto (x, y) —"antes"— que se transforma en el (x', y') —"después"—. Definimos completamente la transformación si podemos expresar las ecuaciones que relacionan los puntos "antes" y "después". Una transformación lineal es aquella en que los puntos "después" pueden expresarse como combinación lineal de las coordenadas del punto "antes"; o sea, las ecuaciones sólo contienen términos proporcionales a x e y, y algún sumando constante —no incluye ni productos de x por y, ni exponentes de x e y distintos de 1, ni otras variables—. Este tipo de ecuaciones pueden escribirse

$$x' = A_{11} \times x + A_{12} \times y + A_{13}$$

$$y' = A_{21} \times x + A_{22} \times y + A_{23}$$

Las constantes A_{ij} son los llamados *coeficientes* de la ecuación. Como puede verse, el resultado de la transformación es una combinación de múltiplos de x, y y de la unidad. Por otra parte, podemos añadir otra ecuación

$$1 = A_{31} \times x + A_{32} \times y + A_{33}$$

Para que se cumpla para cualquier valor de x e y, tendremos que asignar los siguientes valores: $A_{31} = A_{32} = 0$ y $A_{33} = 1$. El añadir esta nueva ecuación podría parecer, a primera vista, un barroquismo inútil; no es este el caso, ya veremos que es de gran utilidad, cuando la utilicemos. Vamos a representar el vector correspondiente a un punto (x, y) (también denominado *vector fila* por razones obvias) en forma de un vector columna de tres dimensiones:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Al expresarlo de esta forma, las tres ecuaciones anteriores pueden expresarse como el producto de una matriz por un vector columna

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

de forma que si representamos la transformación por una matriz, podremos transformar el punto deseado (representado como vector columna) premultiplicándolo por la matriz.

Muchos autores de libros sobre gráficos realizados con ordenador no son partidarios del empleo de vectores columna ya que prefieren extender el vector fila (por ejemplo $(x, y) \rightarrow (x, y, 1)$), y multiplicarlo por la derecha quedando las ecuaciones anteriores en forma matricial como

$$(x', y', 1) = (x, y, 1) \times \begin{pmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{pmatrix}$$

Observe que esta matriz es la *transpuesta* de la matriz de coeficientes de las ecuaciones. Este hecho puede inducir a frecuentes errores entre aquellos que no posean suficiente soltura en el empleo de matrices. Por esta razón en el libro mantenemos la notación en forma de vectores columna. Conforme se familiarice con el trato con las matrices sería una buena idea traducir algunas (o todas) de las transformaciones siguientes a la otra notación. No importa el método que emplee *en tanto que sea consistente*. (La matriz B transpuesta de otra A está dada por $B_{ij} = A_{ji}$ donde $1 \leqslant i, j \leqslant 3$.)

Combinación de transformaciones

Una propiedad interesante de la representación matricial de las transformaciones es que si queremos combinar dos transformaciones (primero una dada por la matriz A y a continuación otra dada por la matriz B, por ejemplo), la transformación total

tiene una matriz asociada $C = B \times A$: observe el orden del producto, la matriz de la primera transformación está *multiplicada por la izquierda* por la de la segunda. El origen de este orden es que la matriz C se utilizará multiplicándola por el vector columna, así que la primera matriz deberá estar a la derecha y la última a la izquierda. (Si hubiéramos empleado el método del vector fila, entonces el producto hubiera aparecido en su *orden natural* de izquierda a derecha; éste es el precio que tenemos que pagar por la identificación de la matriz de transformación con los coeficientes de la ecuación.)

Por tanto, tenemos que presentar la subrutina "mult2", que realiza el producto de dos matrices. El lenguaje BASIC no permite el envío de matrices como parámetros a las subrutinas, de forma que tendremos que inventar un sistema eficiente para copiarlas considerando esta limitación. Supondremos que todos los productos se realizan entre las matrices A y R y el resultado es la matriz B, y después de calcular el producto, copiaremos B de nuevo en la R. Las razones para elegir estos identificadores y realizar la copia final se aclararán conforme vayamos avanzando. También necesitamos una subrutina ("idR2"), que da a R el valor de la matriz identidad. Para formar el producto de una secuencia de matrices, primero hacemos $R = I$ y asociamos cada una de las matrices (de derecha a izquierda) a la matriz A y llamamos a la subrutina "mult2". Al final del proceso, R contendrá el producto de la secuencia de matrices (véase el listado 4.1).

Listado 4.1

```
9100 REM multiplicacion2
9101 REM Datos de entrada A(3,3)
,R(3,3)
9102 REM Datos de salida R(3,3)
9110 FOR I=1 TO 3
9120 FOR J=1 TO 3
9130 LET AR=0
9140 FOR K=1 TO 3
9150 LET AR=AR+A(I,K)*R(K,J)
9160 NEXT K
9170 LET B(I,J)=AR
9180 NEXT J
9190 NEXT I
9200 FOR I=1 TO 3
9210 FOR J=1 TO 3
9220 LET R(I,J)=B(I,J)
9230 NEXT J
9240 NEXT I
9250 RETURN
9300 REM idR2
9302 REM Datos de salida R(3,3)
```

```
9310 FOR I=1 TO 3
9320 FOR J=1 TO 3
9330 LET R(I,J)=0
9340 NEXT J
9350 LET R(I,I)=1
9360 NEXT I
9370 RETURN
```

Todas las transformaciones usuales pueden reducirse a una combinación de tres básicas (que además son lineales): traslación, escala y rotación alrededor del origen. También puede verse que todas las aplicaciones de estas transformaciones que sean válidas, nos proporcionan matrices no singulares. Las subrutinas que siguen generan una matriz denominada A para cada uno de los tres tipos de transformación, de forma que pueden emplearse junto con "mult2" para producir combinaciones de transformaciones.

Traslación

Un punto (x, y) "antes" se desplaza por medio de un vector (TX, TY) al (x', y'). Esto puede representarse por las ecuaciones

$$x' = 1 \times x + 0 \times y + \text{TX}$$

$$y' = 0 \times x + 1 \times y + \text{TY}$$

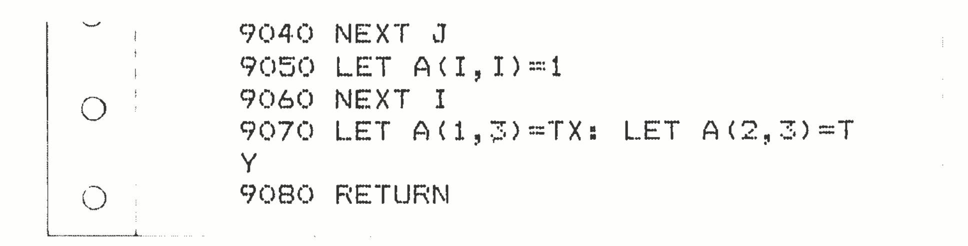
y por tanto la matriz que describe la transformación es

$$\begin{pmatrix} 1 & 0 & \text{TX} \\ 0 & 1 & \text{TY} \\ 0 & 0 & 1 \end{pmatrix}$$

En el listado "tran2" se da una subrutina que genera dicha matriz A, dados los valores TX y TY.

Listado 4.2

```
9000 REM traslacion2
9001 REM Datos de entrada TX,TY
9002 REM Datos de salida A(3,3)
9010 FOR I=1 TO 3
9020 FOR J=1 TO 3
9030 LET A(I,J)=0
```



```
9040 NEXT J
9050 LET A(I,I)=1
9060 NEXT I
9070 LET A(1,3)=TX: LET A(2,3)=T
Y
9080 RETURN
```

Transformación de escala

La coordenada x de un punto del espacio queda multiplicada por un factor de escala SX, y, correspondientemente, la coordenada y por SY. Por tanto

$$x' = SX \times x + 0 \times y + 0$$

$$y' = 0 \times x + SY \times y + 0$$

resultado la matriz

$$\begin{pmatrix} SX & 0 & 0 \\ 0 & SY & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Generalmente tanto SX como SY son positivas, pero si una o ambas son negativas, esta transformación produce, además de un cambio de escala, una reflexión. En particular, si SX = −1 y SY = 1, entonces el punto obtenido es la reflexión del original a través del eje y. La subrutina "escala2" se encarga de crear la matriz A a partir de SX y SY (listado 4.3).

Listado 4.3

```
8900 REM escala2
8901 REM Datos de entrada SX,SY
8902 REM Datos de salida A(3,3)
8910 FOR I=1 TO 3
8920 FOR J=1 TO 3
8930 LET A(I,J)=0
8940 NEXT J
8950 NEXT I
8960 LET A(1,1)=SX: LET A(2,2)=S
Y
8970 LET A(3,3)=1
8980 RETURN
```

Rotación alrededor del origen

Si rotamos un punto en dirección contraria a las agujas del reloj (sentido de giro positivo) un ángulo θ y tomando como centro de giro el origen, tenemos

$$x' = \cos\theta \times x - \sin\theta \times y + 0$$

$$y' = \sin\theta \times x + \cos\theta \times y + 0$$

y la matriz resultante es

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La subrutina "rot2" genera la matriz A dado el ángulo θ (listado 4.4).

Listado 4.4

```
8600 REM rotacion2
8601 REM Datos de entrada THETA
8602 REM Datos de salida A(3,3)
8610 FOR I=1 TO 3
8620 FOR J=1 TO 3
8630 LET A(I,J)=0
8640 NEXT J
8650 NEXT I
8660 LET A(3,3)=1
8670 LET CT=COS THETA: LET ST=SI
N THETA
8680 LET A(1,1)=CT: LET A(2,2)=C
T
8690 LET A(1,2)=-ST: LET A(2,1)=
ST
8700 RETURN
```

Transformaciones inversas

Para cada una de las anteriores transformaciones, existe otra transformación, su inversa, que nos devuelve los puntos transformados a su posición original. Si una transformación está representada por la matriz A, entonces su inversa viene representada por la matriz A^{-1} (inversa de A). No es preciso calcular la matriz inversa

usando el listado 7.5; podemos hallarla directamente por medio de los listados 4.2, 4.3 y 4.4 empleando como parámetros de entrada otros derivados de la transformación directa:

1) Si una traslación viene dada por (TX, TY), su inversa es la traslación (−TX, −TY).

2) Una transformación de escala (SX, SY) tiene como transformación inversa la de escalas (1/SX, 1/SY) (naturalmente, tanto SX como SY deben ser distintas de 0, en caso contrario el espacio bidimensional colapsaría a una recta o un punto).

3) Una rotación de ángulo θ se invierte mediante otra de ángulo $-\theta$.

4) Si la matriz de transformación es el producto de otras elementales de traslación, escala y rotación, por ejemplo $A \times B \times C \times ... \times L \times M \times N$, entonces la transformación inversa es

$$N^{-1} \times M^{-1} \times L^{-1} \times \ldots \times C^{-1} \times B^{-1} \times A^{-1}$$

¡Fíjese en el orden de la multiplicación!

La colocación de un objeto

A menudo queremos colocar un objeto en varios puntos de la pantalla con orientaciones arbitrarias. No sería muy eficiente calcular a mano las coordenadas de los vértices para cada posición del objeto e introducirlos en el programa. En lugar de eso, lo que haremos es definir un sistema arbitrario (pero fijo) de coordenadas, que llamaremos el sistema ABSOLUTO. A continuación daremos las coordenadas de los vértices del objeto en alguna forma sencilla, generalmente con respecto al origen, que llamaremos la posición INICIAL. Los segmentos y áreas del objeto vendrán definidos en función de los vértices. Podemos utilizar las matrices de las transformaciones para mover el objeto desde la posición INICIAL a la ACTUAL en el sistema ABSOLUTO. Las líneas y superficies mantienen su relación con los vértices transformados. La matriz que relaciona las posiciones ACTUAL e INICIAL será denominada, a lo largo del resto del libro, matriz P (a veces le añadiremos una letra como subíndice para identificarla con respecto a otras matrices del mismo tipo). A causa de la restricción de no poder pasar matrices como parámetro a los subprogramas, no generaremos explícitamente la matriz P, sino que se empleará implícitamente para actualizar la matriz R.

Vistas del objeto

Los objetos de una escena pueden moverse con respecto a los ejes del sistema de coordenadas ABSOLUTO. Con respecto al observador de la escena, supondremos que el ojo mira directamente al punto (DX, DY) del sistema ABSOLUTO y su cabeza está inclinada en un ángulo α. Sería conveniente suponer que está mirando

al origen y que la cabeza no está girada (a esta posición la denominaremos OBSERVADA). Por tanto calcularemos otra matriz de forma que el ojo se mueva desde su posición ACTUAL a la OBSERVADA. A esta matriz (ACTUAL → OBSERVADA) la denominaremos matriz Q y constará de una traslación de todos los puntos por un vector $(-\mathrm{DX}, -\mathrm{DY})$, matriz A, y una rotación posterior de un ángulo $-\alpha$, matriz B (¡observe el signo menos!). Por tanto $Q = B \times A$, que es calculada por la subrutina "observación2" (listado 4.5). En general, no calcularemos la matriz Q explícitamente, ya que sólo la usaremos para actualizar la matriz R; sin embargo, si es necesario usar varias veces los valores de la matriz, entonces puede resultar conveniente almacenar Q.

Listado 4.5

```
8200 REM Observacion2
8210 INPUT "(DX,DY) ";DX;",";DY
8220 INPUT "ALFA ";ALPHA
8229 REM Fija la observacion al
punto (DX,DY)
8230 LET TX=-DX: LET TY=-DY
8240 GO SUB tran2: GO SUB mult2
8249 REM Inclina la cabeza un an
gulo ALFA radianes
8250 LET THETA=-ALPHA
8260 GO SUB rot2: GO SUB mult2
8270 RETURN
```

Dibujando un objeto

Combinando la matriz de transformación de INICIAL a ACTUAL, P, con la correspondiente al paso ACTUAL-OBSERVADA, Q, obtenemos la matriz que nos pasa de INICIAL a OBSERVADA, $R = Q \times P$ (siempre designaremos a esta matriz con el nombre R: y no olvide que el resultado de la subrutina "mult2" queda en la matriz R). Transformando todos los vértices de la posición INICIAL por medio de R, y simultáneamente con ellos las líneas y las superficies, logramos que las coordenadas del objeto queden dadas con respecto a un observador que mire de frente hacia el origen del sistema de coordenadas ABSOLUTO, y que, de hecho, está mirando la pantalla. Por tanto, identificaremos el sistema de coordenadas ABSOLUTO con el de la pantalla para hallar la posición de los vértices sobre ella, y a continuación dibujaremos los vértices, líneas y superficies que constituyen el objeto. En la práctica, esto se consigue por medio de una *subrutina de construcción* que emplea la matriz R. Esta subrutina contendrá la información correspondiente a los vértices, líneas y superficies, a continuación la transformará por medio de la matriz R, y quizá

dibuje el objeto finalmente (véase el ejemplo 4.1 más adelante). Como luego veremos, existen situaciones donde es mucho más eficiente almacenar la información relativa a los vértices, líneas y superficies. Por ejemplo, las coordenadas de los vértices pueden almacenarse en los vectores ("arrays" con un subíndice) X e Y, y la información relativa a las líneas en una matriz L ("array" con dos subíndices). Los vértices pueden guardarse tanto en la posición INICIAL, como ACTUAL u OBSERVADA —realmente depende el contexto del programa elegir una posición u otra—. Este método del paso de la posición INICIAL a la ACTUAL y de ella a la OBSERVADA, nos permitirá dibujar una serie de escenas en movimiento —los objetos pueden desplazarse con respecto a los ejes ABSOLUTOS, y a ellos mismos; mientras que, simultáneamente, el observador puede ir cambiando su punto y ángulo de visión independientemente del movimiento del objeto—. De todas formas, aunque el campo de posibilidades es muy amplio, empezaremos por el caso más sencillo: una escena fija.

Dibujos complicados. El método modular

Podemos dibujar alguna escena que contenga varios objetos similares. No hay necesidad de escribir una subrutina para cada uno de los objetos, lo que podemos hacer es calcular cada vez una nueva matriz de paso de INICIAL a OBSERVADA y meterla en la misma subrutina. Naturalmente, tendremos que escribir una subrutina distinta de construcción para cada tipo distinto de objetos. El dibujo final se realizará por medio de la ejecución del subprograma "escena2", que será llamada desde un programa principal estándar (listado 4.6). Este programa principal define simplemente las etiquetas de varios subprogramas, declara las variables dimensionadas ("arrays"), centra la zona gráfica, mediante petición de los valores HORIZ y VERT, y, por último, llama a "escena2".

Listado 4.6

```
 100 REM programa principal
 109 REM Define los identificado
res para las rutinas de iniciali
zacion y dibujo 2-D
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400: LET clip=8400
 120 LET rot2=8600: LET angle=88
00: LET scale2=8900: LET tran2=9
000: LET mult2=9100: LET idR2=93
00
 130 LET scene2=6000: LET look2=
8200
```

```
 140 INPUT "HORIZ",HORIZ,"VERT",
VERT
 150 GO SUB start
 160 LET XMOVE=HORIZ*0.5: LET YM
OVE=VERT*0.5
 170 GO SUB setorigin
 179 REM Define la escena
 180 GO SUB scene2
 190 STOP
```

"escena2" llama en primer lugar a "observación2" y genera la matriz Q; si tenemos que dibujar más de un objeto, iremos almacenándolos. Para cada objeto individual ("módulo") calculamos una matriz P y llamamos a la subrutina de construcción requerida usando la matriz $R = Q \times P$. Todas las piezas forman al final el dibujo completo. Para diferenciar las matrices P y R de distintos objetos, le añadiremos un subíndice cuando sea necesario.

Este sistema modular (construimos las piezas y luego las ensamblamos) puede que no sea el más eficiente para definir y realizar un dibujo; no obstante, desde el punto de vista de nuestra experiencia, es el que mejores resultados proporciona a los que empiezan, ya que les permite realizar las preguntas adecuadas para resolver el problema de la construcción de una determinada escena. Aún más, cuando trabajemos con imágenes en movimiento, este método simplificará los problemas en escenas donde, no sólo hay objetos que se mueven entre sí, sino que también el observador está en movimiento.

Por supuesto, si el observador mira de frente a la pantalla, entonces la matriz Q puede sustituirse por una llamada a "fijaorigen", que cambia el sistema de coordenadas de la pantalla. Si además el ojo del observador mira al origen de coordenadas, entonces Q es la matriz identidad I; y por tanto la subrutina "observación2" no incluye ningún efecto y puede ignorarse. Por nuestra parte no supondremos eso y trabajaremos con la situación más general: una de las mejores cosas que puede hacer el lector es *digerir* los programas propuestos y tratar de optimizarlos para algunos casos particulares. Nuestra idea es explicar estos conceptos de la forma más general y directa posible, incluso a expensas de la eficiencia y velocidad del programa. El lector puede volver a estos programas cuando comprenda y domine las ideas sobre las transformaciones del espacio. Más adelante daremos algunas pistas sobre cómo hacer estos cambios, pero de momento, si nos dedicáramos a la optimización sólo, estaríamos metiendo arena en un engranaje que empieza a girar.

Sin embargo, la razón de más peso para emplear este método modular se verá cuando dibujemos objetos tridimensionales. Definiremos las construcciones en tres dimensiones como una extensión de las ideas anteriores, y es absolutamente necesario tener un dominio completo de las transformaciones bidimensionales antes de continuar en espacios con más dimensiones.

Ejemplo 4.1

Sea una sencilla *nave espacial* en su posición INICIAL apuntando hacia la dirección de las x positivas (o sea, formando un ángulo de 0 radianes con el semieje positivo x). La nave está definida por cinco segmentos que unen, por este orden, los puntos $(3, 0), (0, 0), (-1, 1), (2, 0), (-1, -1)$ y de vuelta al $(0, 0)$. Vea la figura 4.1; que es la nave dibujada sobre una pantalla de 5 unidades por 3, donde la matriz INICIAL-ACTUAL es la identidad, y la ACTUAL-OBSERVADA es tal que el observador mira de frente al punto $(1, 0)$. El listado 4.7 nos da la subrutina "escena2" necesaria para colocar el objeto en posición, y el listado 4.8 ("nave") es la subrutina requerida para construir la nave. Observe que "nave", que emplea la matriz R para transformar los vértices (y por tanto el objeto) a la posición OBSERVADA, no almacena los valores de los vértices en esta posición en una base de datos permanente. En lugar de ello los valores se guardan en los "arrays" X e Y hasta que la subrutina devuelva el control al programa que la llamó, y si vuelve a ser ejecutada para dibujar otra nave espacial, entonces estos vectores son empleados para guardar los nuevos datos.

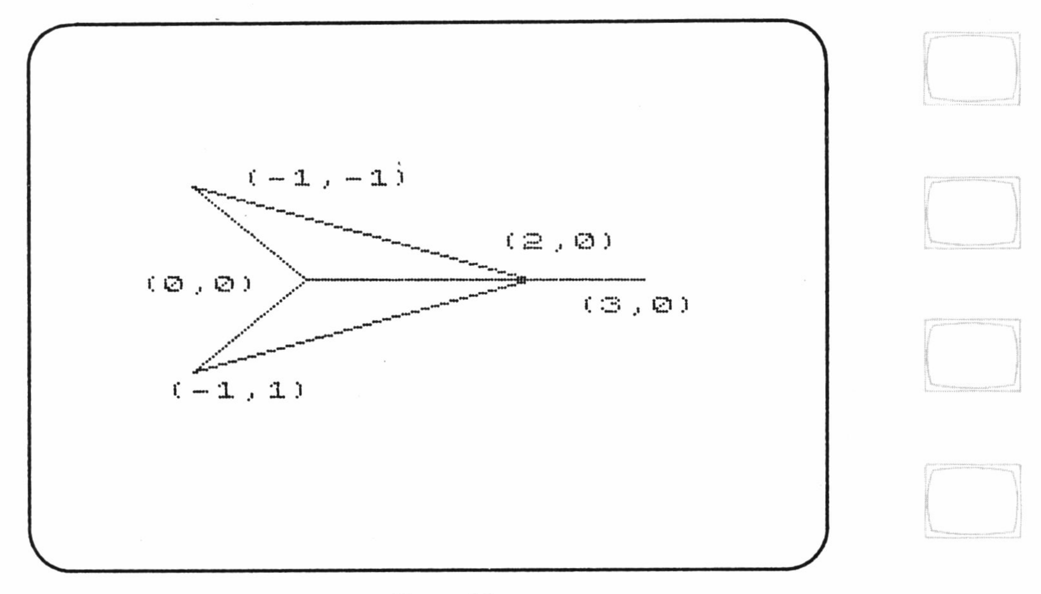


Figura 4.1

Listado 4.7

```
6000  REM escena2/observacion2;
nave (no almacenada)
6010 DIM X(6): DIM Y(6)
6020 DIM A(3,3): DIM B(3,3): DIM
 R(3,3)
```

```
6030 LET ship = 6500
6039 REM coloca al observador
6040 GO SUB idR2: GO SUB look2
6049 REM define y dibuja el obje
to
6050 GO SUB ship
6060 RETURN
```

Listado 4.8

```
6500 REM nave/no almacenada
6509 REM IN: R(3,3)
6510 DATA 3,0,0,0,-1,1,2,0,-1,-1
,0,0
6520 RESTORE ship
6530 FOR I = 1  TO 6
6539 REM lee las coordenadas del
 objeto INICIAL
6540 READ XX,YY
6549 REM mueve el objeto a la po
sicion OBSERVADA
6550 LET X(I) = XX*R(1,1) + YY*R
(1,2) + R(1,3)
6560 LET Y(I) = XX*R(2,1) + YY*R
(2,2) + R(2,3)
6570 NEXT I
6579 REM une los vertices por or
den
6580 LET XPT = X(1): LET YPT = Y
(1): GO SUB moveto
6590  FOR I = 2 TO 6
6600 LET XPT = X(I): LET YPT = Y
(I): GO SUB lineto
6610 NEXT I
6620 RETURN
```

Ejemplo 4.2

Suponga que deseamos dibujar la figura 4.2, que incluye cuatro naves espaciales llamadas a), b), c) y d) en una pantalla de 60 × 40 unidades. Por sencillez supondremos que Q es la matriz identidad (por esta vez, pase), de forma que el observador mira

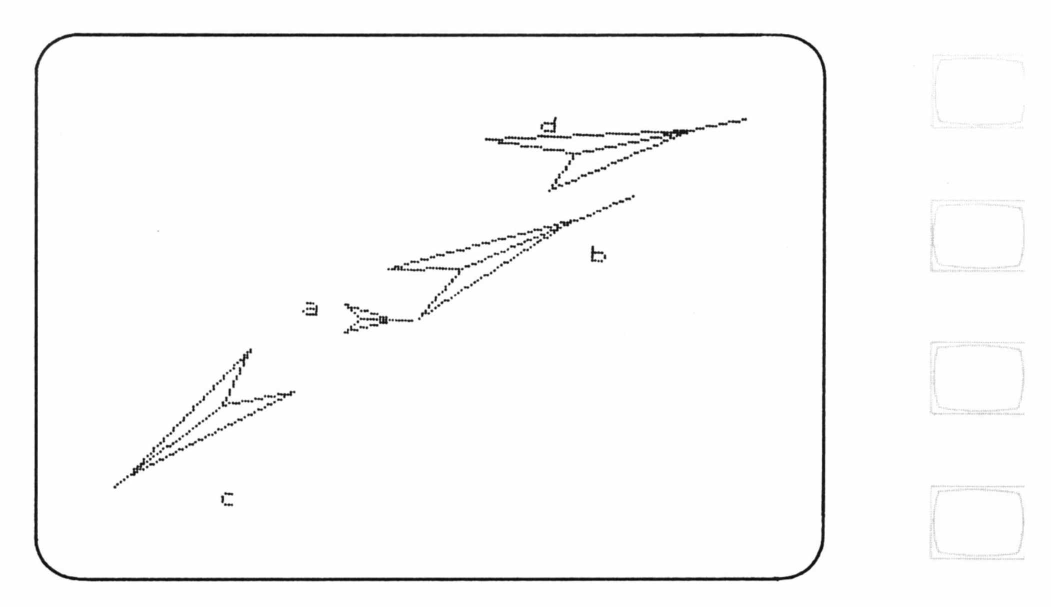


Figura 4.2

de frente y su ojo apunta hacia el origen INICIAL. La nave a) está colocada en su posición INICIAL; esto es, $R_a = I$, mientras que la b) se ha movido desde la posición INICIAL a la ACTUAL por medio de las siguientes transformaciones:

1) cambio de escala con SX = 4 y SY = 2, resultando la matriz A.

2) rotación de la figura $\pi/6$ radianes, matriz B.

3) traslación con TX = 6 y TY = 4, matriz C.

$$A = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 0 & 6 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

La transformación completa está dada por $R_b = Q \times P_b = I \times P_b = P_b = C \times$ $\times B \times A$ (observe el orden en la multiplicación de las matrices, y que el subíndice indica que nos referimos a la nave b).

Si hacemos el producto en el orden $A \times B \times C$ (dando la matriz P_d), entonces obtenemos

$$P_b = \begin{pmatrix} 2\sqrt{3} & -1 & 6 \\ 2 & \sqrt{3} & 4 \\ 0 & 0 & 1 \end{pmatrix} \quad P_d = \begin{pmatrix} 2\sqrt{3} & -2 & 12\sqrt{3}-8 \\ 1 & \sqrt{3} & 4\sqrt{3}+6 \\ 0 & 0 & 1 \end{pmatrix}$$

que son, obviamente, dos transformaciones distintas. La matriz $R_d = Q \times P_d = I \times P_d$ produce la nave d). Observe cómo esta nave es asimétrica. Sea muy cauteloso con los cambios de escala —¡recuerde que se definen alrededor del origen y producirán distorsiones en la forma de un objeto que se haya movido del origen!

Para ilustrar aún más este ejemplo veremos cómo se calcula la posición ACTUAL de la nave b) en la pantalla colocando las coordenadas en forma de vector columna y premultiplicándolo por la matriz $R_b = I \times P_b$; por ejemplo,

$$\begin{pmatrix} 2\sqrt{3} & -1 & 6 \\ 2 & \sqrt{3} & 4 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 6\sqrt{3}+6 \\ 10 \\ 1 \end{pmatrix} \quad \text{etc.}$$

Cuando volvemos a la forma vectorial normal vemos que los cinco vértices han sido convertidos en los $(6\sqrt{3}+6, 10), (6, 4), (5-2\sqrt{3}, \sqrt{3}+2), (4\sqrt{3}+6, 8)$ y $(7 - 2\sqrt{3}, 2-\sqrt{3})$ respectivamente.

La nave c) es la b) reflejada sobre la recta $3y = -4x - 9$. Esta recta corta al eje y en el punto $(0, 3)$ y forma un ángulo $\alpha = \cos^{-1}(-3/5) = \text{sen}^{-1}(4/5) = \tan^{-1}(-3/4)$ con el semieje x positivo. Si movemos el espacio según el vector $(0, 3)$, matriz D, esta recta pasará por el origen. Más aún, si giramos el espacio un ángulo $-\alpha$, matriz E, la recta coincide con el eje x. La matriz F refleja la nave según el eje x, E^{-1} devuelve a la recta la inclinación de ángulo α, y por último D^{-1} coloca la recta en su posición original. La matriz $G = D^{-1} \times E^{-1} \times F \times E \times D$ nos dará los vértices ACTUALES de la nave b) reflejados en la recta $3y = -4x - 9$, y $R_c = I \times P_c = G \times P_b$ puede emplearse para dibujar la nave c). Resumiendo, empleamos la matriz P_b para mover la nave a la posición b) y a continuación por medio de G la colocamos en la posición c).

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} -3/5 & 4/5 & 0 \\ -4/5 & -3/5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad F = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

y

$$R_c = \frac{1}{25}\begin{pmatrix} -48 - 14\sqrt{3} & 7 - 25\sqrt{3} & -210 \\ 14 - 48\sqrt{3} & 24 + 7\sqrt{3} & -170 \\ 0 & 0 & 25 \end{pmatrix}$$

La figura 4.2 se dibuja con la nueva "escena2" (listado 4.9): observe que esta "escena2" no llama a "observación2", ya que el ojo está mirando al origen y de frente. El programa principal y la subrutina "nave", así como todas las otras subrutinas gráficas, se mantienen sin cambios.

Listado 4.9

```
6000  REM escena2/sin observacio
n2; 4 naves (no almacenadas)
6010 DIM X(6): DIM Y(6)
6020 DIM A(3,3): DIM B(3,3): DIM
 R(3,3)
6030 LET ship = 6500
6034  REM nave a).
6039 REM OBSERVADA= ACTUAL, no e
s necesario llamar a "observacio
n2"
6040 GO SUB idR2: GO SUB ship
6049  REM nave b).
6050 LET SX = 4: LET SY = 2
6060 GO SUB scale2: GO SUB mult2
6070 LET THETA = PI/6
6080 GO SUB rot2: GO SUB mult2
6090 LET TX = 6: LET TY = 4
6100 GO SUB tran2: GO SUB mult2
6110 GO SUB ship
6119  REM nave c).
6120 LET AX = -3: LET AY = 4
6130 GO SUB angle
6140 LET TX = 0: LET TY = 3
6150 GO SUB tran2: GO SUB mult2
6160 LET THETA = -THETA
6170 GO SUB rot2: GO SUB mult2
6180 LET SX = 1: LET SY = -1
6190 GO SUB scale2: GO SUB mult2
6200 LET THETA = -THETA
6210 GO SUB rot2: GO SUB mult2
6220 LET TX = 0: LET TY = -3
```

```
6230 GO SUB tran2: GO SUB mult2
6240 GO SUB ship
6249  REM nave d).
6250 GO SUB idR2
6260 LET TX = 6: LET TY = 4
6270 GO SUB tran2: GO SUB mult2
6280 LET THETA = PI/6
6290 GO SUB rot2: GO SUB mult2
6300 LET SX = 4: LET SY = 2
6310 GO SUB scale2: GO SUB mult2
6320 GO SUB ship
6330 RETURN
```

Ejercicio 4.1

Con el fin de que se convenza de que este programa puede ser utilizado en el caso más general, debería ejecutarlo con valores DX, DY o α distintos de 0, de forma que la matriz Q (ACTUAL-OBSERVADA) no sea la matriz identidad. Su subrutina “escena2” debería llamar a “observación2” para calcular Q, que debe almacenarse. A continuación se calcula la matriz P (INICIAL-ACTUAL) para cada objeto (que “mult2” colocará en R), se premultiplica por Q (copiándola en A para su uso por “mult2”) y finalmente se llama a la subrutina de construcción con la matriz $R = Q \times P$. Asegúrese de que la subrutina “trazalíneas” contiene la opción “recorte” o se encontrará con que el programa se detiene cuando intente dibujar fuera del rectángulo del área gráfica.

Ejercicio 4.2

Utilice las subrutinas anteriores para dibujar diagramas similares al de la figura 4.2, pero donde el número, la posición y la dirección de las naves sean datos a leer desde el teclado. Puede escribir subrutinas para dibujar objetos más elevados; elegimos un ejemplo muy sencillo para que los algoritmos no quedarán enmascarados por la complejidad de los objetos. El método anterior es aplicable a tantos vértices y líneas como el Spectrum pueda manejar con sus limitaciones de tiempo y espacio. Los objetos no tienen por qué ser solamente unos trazos, puede dibujar también áreas coloreadas (polígonos limitados por los vértices transformados).

Ejercicio 4.3

Empleando bucles (FOR ... NEXT) en el programa podemos dibujar secuencias ordenadas de objetos; por ejemplo, pueden tener la misma orientación pero con puntos de referencia (el origen en la posición INICIAL) igualmente espaciados a lo

largo de la línea $\boldsymbol{p} + \mu\boldsymbol{q}$. Podemos crear un bucle con un parámetro de índice μ y dibujar una nave para cada vuelta del bucle. Para cada valor de μ podemos cambiar los parámetros de la traslación de una forma continua dentro del bucle (usando μ, $\boldsymbol{p}$ y $\boldsymbol{q}$). Los nuevos valores de estos parámetros se usan para calcular la correspondiente matriz INICIAL-ACTUAL, y el objeto cambia a la nueva posición ACTUAL. Se emplea como siempre la matriz $R = Q \times P = I \times P$ para observar y dibujar cada objeto en la pantalla. Con estas ideas, construya un conjunto de *formaciones de combate* con la nave espacial de los ejemplos anteriores.

Uso eficiente de las matrices

Es obvio que en cualquier combinación de transformaciones que empleemos, la tercera fila de cualquier matriz será siempre (0 0 1). Si trabajamos con las otras dos filas solamente, las subrutinas serán mucho más rápidas. Mantenemos todavía matrices de 3 × 3 en lugar de 2 × 3 (que es lo que realmente necesitamos), ya que hemos escrito otras subrutinas que suponen que las matrices son 3 × 3. Intentar reDIMensionar las variables nos conduciría a errores de límites de dimensión en las subrutinas primeras —el coste de tres números reales innecesarios por matriz es un precio a pagar muy pequeño para correr el riesgo de sufrir numerosos errores—. Nótese también que, cuando DIMensionamos una variable, ésta es puesta a cero inmediatamente (consúltese el capítulo correspondiente a las variables indexadas (*arrays*) del Manual de BASIC del Spectrum). A continuación reescribimos los listados 4.1, 4.2, 4.2 y 4.4 con los nuevos nombres 4.1a, 4.2a, 4.3a y 4.4a, respectivamente, para aplicar estas posibilidades.

Listado 4.1a

```
9100 REM multiplicacion2
9101 REM Datos de entrada A(3,3)
,R(3,3)
9102 REM Datos de salida R(3,3)
9110 FOR I=1 TO 2
9120 FOR J=1 TO 3
9130 LET B(I,J)=A(I,1)*R(1,J)+A(
I,2)*R(2,J)
9140 NEXT J
9150 LET B(I,3)=B(I,3)+A(I,3)
9160 NEXT I
9170 FOR J=1 TO 3
9180 LET R(1,J)=B(1,J): LET R(2,
J)=B(2,J)
9190 NEXT J
9200 RETURN
```

```
9300 REM identidadR2
9302 REM Datos de salida R(3,3)
9310 DIM R(3,3)
9320 LET R(1,1)=1: LET R(2,2)=1:
 LET R(3,3)=1
9330 RETURN
```

Listado 4.2a

```
9000 REM translacion2
9001 REM Datos de entrada TX,TY
9002 REM Datos de salida A(3,3)
9010 DIM A(3,3)
9020 LET A(1,1)=1: LET A(2,2)=1
9030 LET A(1,3)=TX: LET A(2,3)=T
Y
9040 RETURN
```

Listado 4.3a

```
8900 REM escala2
8901 REM Datos de entrada SX,SY
8902 REM Datos de salida A(3,3)
8910 DIM A(3,3)
8920 LET A(1,1)=SX: LET A(2,2)=S
Y
8930 RETURN
```

Listado 4.4a

```
8600 REM rotacion2
8601 REM Datos de entrada THETA
8602 REM Datos de salida A(3,3)
8610 DIM A(3,3)
8620 LET CT=COS THETA: LET ST=SI
N THETA
```

```
8630 LET A(1,1)=CT: LET A(2,2)=C
T
8640 LET A(1,2)=-ST: LET A(2,1)=
ST
8650 RETURN
```

La construcción de la figura 4.2 puede parecer algo académica al ser elegida la posición de los objetos de una forma arbitraria. Por el contrario, en la mayoría de los diagramas las posiciones de los objetos están perfectamente definidas, estando los valores fijados implícitamente por el diagrama requerido. Vea el ejemplo siguiente.

Ejemplo 4.3

Escriba un programa que dibuje una elipse de eje mayor A, eje menor B y centrada en el punto (CX, CY). El eje mayor forma un ángulo θ con el semieje x positivo. El orden de las transformaciones es importante: primero gire y luego desplace. Si lo que queremos es dibujar elipses con el eje mayor horizontal, entonces no tenemos por qué emplear matrices, podemos seguir con la subrutina del ejercicio 2.5 con ideas similares a las del listado 2.11a. El listado 4.10 ofrece una subrutina "escena2" que lee los datos necesarios para la construcción de la elipse, calcula la matriz INICIAL-OBSERVADA y llama a la subrutina de construcción "elipse" que dibuja la elipse.

Listado 4.10

```
6000 REM escena2/elipse
6010 DIM A(3,3): DIM B(3,3): DIM
 R(3,3)
6020 LET ellipse=6500
6030 INPUT "(CX,CY) ";CX;",";CY,
,"A ";A;",B ";B;",TETA ";THETA
6040 LET THETA=-THETA
6049 REM Elipse centrada en (CX,
CY) e inclinada un angulo TETA
6050 GO SUB idR2
6060 GO SUB rot2: GO SUB mult2
6070 LET TX=CX: LET TY=CY
6080 GO SUB tran2: GO SUB mult2
6090 GO SUB ellipse
```

```
6100 RETURN
6500 REM elipse
6501 REM Datos de entrada A,B,R(
3,3)
6509 REM Eje mayor de la elipse
A, eje menor B
6510 LET XPT=A*R(1,1)+R(1,3)
6520 LET YPT=A*R(2,1)+R(2,3)
6530 GO SUB moveto
6540 LET ALPHA=0: LET ADIFF=PI/1
00
6549 REM Calcula los puntos (XPT
,YPT) de la elipse en la posicio
nnobservada
6550 FOR I=1 TO 200
6560 LET ALPHA=ALPHA+ADIFF
6570 LET AA=A*COS ALPHA: LET BB=
B*SIN ALPHA
6580 LET XPT=AA*R(1,1)+BB*R(1,2)
+R(1,3)
6590 LET YPT=AA*R(2,1)+BB*R(2,2)
+R(2,3)
6600 GO SUB lineto
6610 NEXT I
6620 RETURN
```

Ejercicio 4.4

Escriba una subrutina para dibujar un objeto individual (en este caso un *astroide* como el mostrado en la figura 4.3a) capaz de ser transformado por medio de matrices. A continuación dibuje combinaciones del objeto empleando las técnicas de transformación matricial descritas anteriormente (como, por ejemplo, en la figura 4.3b). Un astroide es una curva cerrada cuya expresión en forma paramétrica es ($R \cos^3 \theta / \operatorname{sen}^3 \theta$) donde $0 \leqslant \theta \leqslant 2\pi$, siendo R el radio (la distancia máxima desde el centro de la figura). Los parámetros que precisa esta subrutina son el radio del astroide y la matriz de transformación. La figura 4.3b es la combinación de gran cantidad de dos tipos de astroides: el primero de radio unidad y sin girar, y el segundo de radio $\sqrt{2}$ y rotado $\pi/4$ radianes.

Ejercicio 4.5

Haga experimentos con estas técnicas matriciales. Escriba una subrutina que genere la matriz necesaria para girar puntos en el espacio un ángulo θ alrededor de un punto arbitrario (x, y) del espacio (no necesariamente el origen). Escriba también

otra subrutina que genere la matriz necesaria para reflejar un punto sobre la recta $ay = bx + c$. (Utilice las ideas dadas en el ejemplo 4.2 para el trazado de la nave c).)

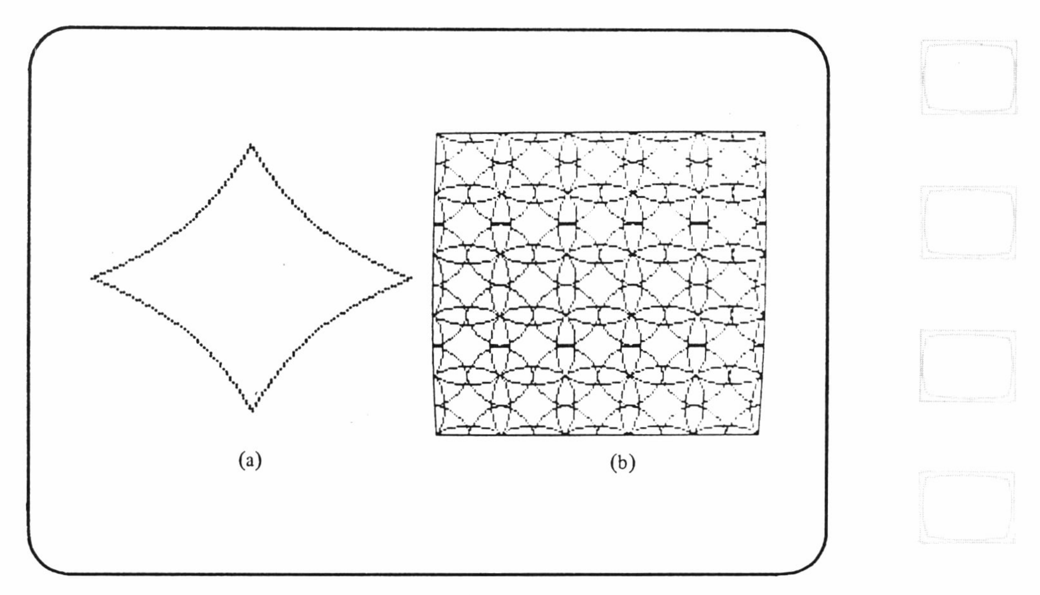


Figura 4.3.

Almacenamiento de información relativa a las escenas

Como mencionamos anteriormente, nos encontraremos en ciertas situaciones donde precisaremos almacenar toda la información relativa a una escena en una gran base de datos. Constará de los vectores X e Y, de longitud mayor o igual a NOV, el número total de vértices a guardar. (Estos vértices pueden almacenarse tanto en la posición INICIAL, como en la ACTUAL o en la OBSERVADA: depende del contexto del problema.) También necesitamos guardar la información relativa a las líneas en una matriz con dos dimensiones L cuyo primer índice es 1 ó 2, y el segundo un valor mayor o igual a NOL, el número total de líneas de la escena. La I-ésima línea une los vértices dados por L(1,I) con L(2,I): por tanto la información es independiente de la posición, simplemente indica cuáles vértices están unidos por el segmento I-ésimo. NOV y NOL toman sus valores iniciales en la subrutina "escena2" y van siendo incrementados por las subrutinas de construcción.

Siguiendo este método ya no es necesario diseñar y escribir las subrutinas de construcción que dibujen las líneas, sólo las empleamos para crear la base de datos de líneas, vértices, etc. (transformada por medio de la matriz *R*). Después de que "escena2" haya montado la escena completa en memoria, llama a otra subrutina ("dibujo"), que dibuja la imagen final. La subrutina "escena2" será muy parecida a la mencionada anteriormente; por ejemplo, el subprograma para dibujar la figura 4.2 por este nuevo sistema es el mismo del listado 4.9, con los tres pequeños cambios siguientes:

```
6010 DIM X(20): DIM Y(20): DIM L
(2,20)

6030 LET NOV = 0: LET NOL = 0: L
ET ship = 6500: LET drawit = 700
0

6330 GO SUB drawit: RETURN
```

Esta subrutina junto con el listado 4.11 (subrutina de construcción de la nave, que sólo inicializa los datos) y la subrutina "dibujo" realizarán el gráfico de la figura 4.2 por este método de almacenamiento.

Listado 4.11

```
6500 REM nave/almacen. datos
6501 REM Datos de entrada NOV,NO
L,R(3,3),X(NOV),Y(NOV)
6502 REM Datos de salida NOV,NOL
,X(NOV),Y(NOV),L(2,NOL)
6510 DATA 3,0,0,0,-1,1,2,0,-1,-1
,1,2,2,3,3,4,4,5,5,2
6520 RESTORE ship
6530 LET NV=NOV
6531 REM Lee los vertices y los
coloca en posicion usando la mat
riz R
6540 FOR I=1 TO 5
6550 READ XX,YY
6560 LET NOV=NOV+1
6570 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+R(1,3)
6580 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+R(2,3)
6590 NEXT I
6591 REM Lee y almacena la linea
 de informacion
6600 FOR I=1 TO 5
6610 READ L1,L2
6620 LET NOL=NOL+1
6630 LET L(1,NOL)=L1+NV: LET L(2
,NOL)=L2+NV
```

```
6640 NEXT I
6650 RETURN

7000 REM dibujo
7001 REM Datos de entrada NOL,X(
NOV),Y(NOV),L(2,NOL)
7009 REM Traza rectas uniendo pa
res de vertices
7010 FOR I=1 TO NOL
7020 LET L1=L(1,I): LET L2=L(2,I
)
7030 LET XPT=X(L1): LET YPT=Y(L1
): GO SUB moveto
7040 LET XPT=X(L2): LET YPT=Y(L2
): GO SUB lineto
7050 NEXT I
7060 RETURN
```

Suponga que deseamos representar distintas vistas de la misma escena (de nuevo emplearemos la figura 4.2 como ejemplo); esto es, las mismas matrices *P* (INICIAL → ACTUAL), pero distintas matrices *Q* (ACTUAL → OBSERVADA). La solución evidente es crear una base de datos para la escena que contenga los vértices en la posición ACTUAL. Para cada nueva posición OBSERVADA, calcularemos la matriz *Q* correspondiente y la mandaremos a otra subrutina "dibujo" (vea el listado 4.12, distinto del 4.11), que transfiere cada vértice desde su posición ACTUAL a la OBSERVADA por medio de *Q*, los almacena en los vectores V y W, y los recupera cuando los necesite para dibujarlos. Cuando use este método para construir distintas vistas de la figura 4.2, solamente las subrutinas "escena2" y "dibujo" difieren de las anteriores, y aun así, muy ligeramente. Las presentamos en el listado 4.12.

Listado 4.12

```
6000 REM escena2/ observacion2 v
ariable; 4 naves (almacenadas)
6009 REM Construye 4 naves almac
enadas en la posicion ACTUAL
6010 DIM X(20): DIM Y(20): DIM V
(20): DIM W(20): DIM L(2,20)
6020 DIM A(3,3): DIM B(3,3): DIM
 R(3,3)
6030 LET ship=6500: LET drawit=7
000
6039 REM Nave a)
```

```
6040 LET NOV=0: LET NOL=0: GO SU
B idR2: GO SUB ship
6049 REM Nave b)
6050 LET SX=4: LET SY=2
6060 GO SUB scale2: GO SUB mult2
6070 LET THETA=PI/6
6080 GO SUB rot2: GO SUB mult2
6090 LET TX=6: LET TY=4
6100 GO SUB tran2: GO SUB mult2
6110 GO SUB ship
6119 REM Nave c)
6120 LET AX=-3: LET AY=4
6130 GO SUB angle
6140 LET TX=0: LET TY=3
6150 GO SUB tran2: GO SUB mult2
6160 LET THETA=-THETA
6170 GO SUB rot2: GO SUB mult2
6180 LET SX=1: LET SY=-1
6190 GO SUB scale2: GO SUB mult2
6200 LET THETA=-THETA
6210 GO SUB rot2: GO SUB mult2
6220 LET TX=0: LET TY=-3
6230 GO SUB tran2: GO SUB mult2
6240 GO SUB ship
6249 REM Nave d)
6250 GO SUB idR2
6260 LET TX=6: LET TY=4
6270 GO SUB tran2: GO SUB mult2
6280 LET THETA=PI/6
6290 GO SUB rot2: GO SUB mult2
6300 LET SX=4: LET SY=2
6310 GO SUB scale2: GO SUB mult2
6320 GO SUB ship
6329 REM Bucle para los puntos d
e observacion
6330 GO SUB idR2: GO SUB look2
6340 CLS : GO SUB drawit
6350 GO TO 6330
6360 RETURN

7000 REM dibujo/transforma posic
ion ACTUAL a OBSERVADA y dibuja
7001 REM Datos de entrada NOV,NO
L,R(3,3),X(NOV),L(2,NOL)
7009 REM Transforma los vertices
```

```
 de la posicion ACTUAL a la OBSE
RVADA
7010 FOR I=1 TO NOV
7020 LET V(I)=X(I)*R(1,1)+Y(I)*R
(1,2)+R(1,3)
7030 LET W(I)=X(I)*R(2,1)+Y(I)*R
(2,2)+R(2,3)
7040 NEXT I
7049 REM Traza rectas uniendo pa
res de vertices
7050 FOR I=1 TO NOL
7060 LET L1=L(1,I): LET L2=L(2,I
)
7070 LET XPT=V(L1): LET YPT=W(L1
): GO SUB moveto
7080 LET XPT=V(L2): LET YPT=W(L2
): GO SUB lineto
7090 NEXT I
7100 RETURN
```

Ejercicio 4.6

Construya una escena en movimiento. En cada nueva vista las naves se moverán unas con respecto a otras de alguna manera perfectamente definida. El observador debería moverse de alguna forma sencilla; por ejemplo, el ojo comienza mirando al origen, cinco vistas más tarde está mirando al punto (10, 10), y en cada vista la cabeza se gira un ángulo de 0.1 radianes con respecto a la posición anterior. No es necesario que introduzca los valores de (DX, DY) y ALPHA por el teclado en "observación3", en lugar de ello, el programa debería calcularlos. Después de que haya leído el capítulo 13 podrá colocar en memoria estas cinco figuras y recuperarlas como una "película" (si tiene la máquina de 48K).

Ejercicio 4.7

Construya una escena que sea una vista esquemática de una habitación de su casa —con bocetos bidimensionales de mesas, sillas, etc., colocados en la habitación—. Cada tipo de objeto tendrá su propia subrutina de construcción, y el subprograma "escena2" debería leer los datos para colocarlos en la habitación. Una vez la escena esté completa, genere varias vistas, mirando desde varios puntos y con distintas orientaciones.

También puede dibujar un mapa con segmentos rectilíneos, y de nuevo verlo desde varias orientaciones. ¡El número de posibles escenas que puede elegir es enorme!

Como empleamos la opción "recorte" en "trazalíneas" podemos elegir valores pequeños para HORIZ y VERT, que producen el efecto de ampliar (como con un *zoom*) los detalles de una escena, mientras que todas las líneas que se salgan de la pantalla serán eliminadas.

Programas completos

Agrupamos los listados 3.4, "ángulo" *(angle);* 4.1a, "multiplicación2" e "idR2" *(mult2* e *idR2);* 4.2a, "traslación" *(tran2);* 4.3a, "escala2" *(scale2);* 4.4a, "rotación2" *(rot2);* 4.5, "observación2" *(look2)* y 4.6, "programa principal" *(main program)* bajo el título "rut2".

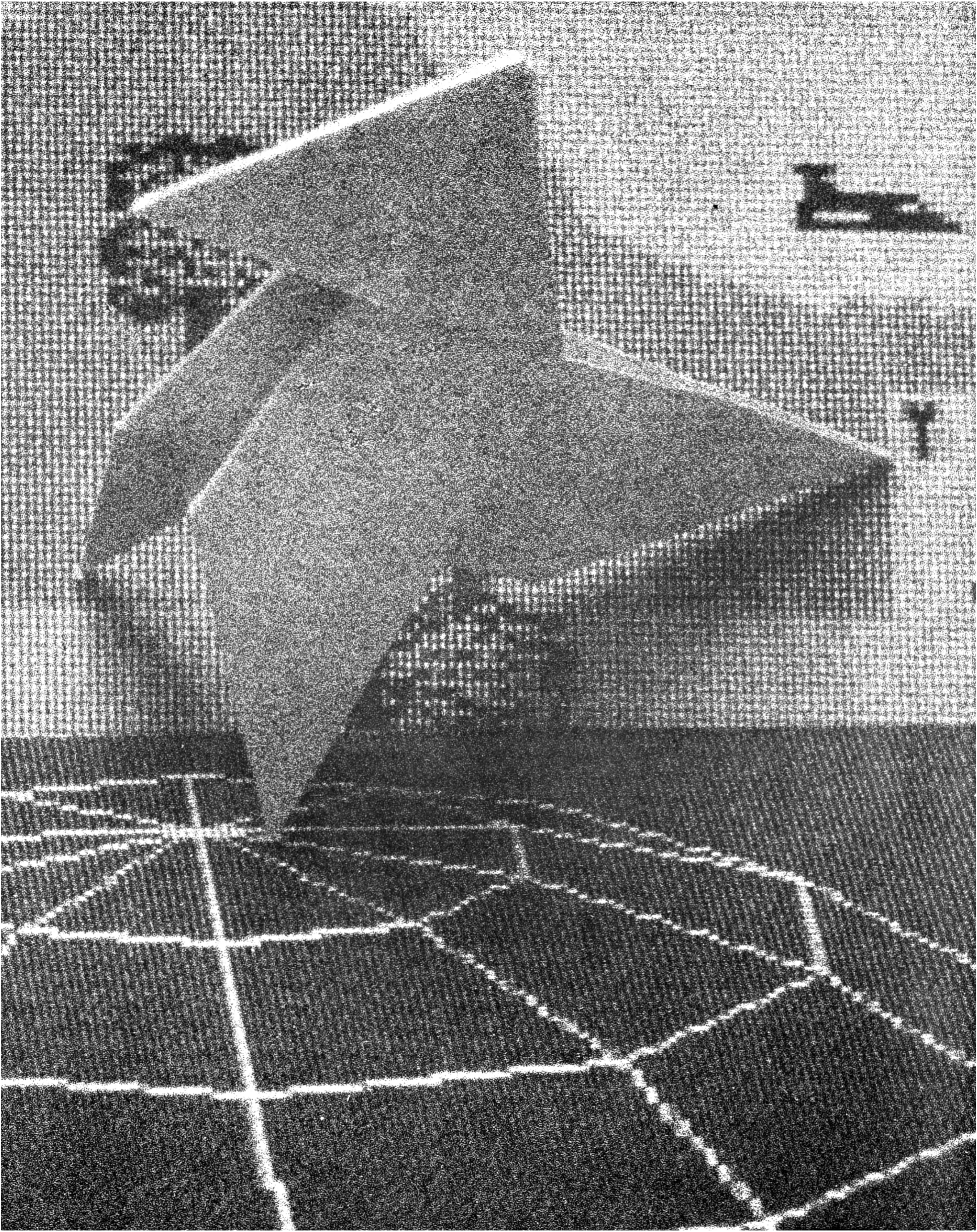
I. "rut1", "rut2", listados 4.7, "escena2" *(scene2)* y 4.8, "nave" *(ship)*. Datos requeridos: HORIZ, VERT, DX, DY y ALPHA. Pruebe con 8, 5, 1, 1, 0.5. Mantenga fijos cuatro de éstos y haga pequeños cambios, de una forma sistemática, en el otro.

II. "rut1", "rut2", listado 4.9, "escena2" y 4.8, "nave". Datos requeridos: HORIZ y VERT. Pruebe con 30, 20; 200, 200; 200, 150.

III. "rut1", "rut2", listados 4.10, "escena2/elipse". Datos requeridos: HORIZ, VERT, CX, A, B, THETA. Pruebe con 30, 20, 0, 0, 12, 9, 0. De nuevo deje fijos todos menos uno y cámbielo de una forma sistemática.

IV. "rut1", "rut2", listados 4.9 ("escena2" corregido según el texto) y 4.11, "nave" y "dibujo". Datos requeridos: como para II.

V. "rut1", "rut2", listados 4.11 ("nave", pero sin "dibujo") y 4.12, "escena2" y "dibujo". Datos requeridos: HORIZ, VERT, DX, DY, ALPHA. Pruebe con 60, 40, 0, 0, 0. Cambie cada uno de los valores por turno de una forma sistemática.



5
Gráficos con caracteres en el ZX Spectrum

En el capítulo primero (listado 1.2) vimos cómo podía generarse un pequeño bloque o carácter, a través de ocho números binarios. Este bloque de 8 × 8 *pixels* es denominado un *bloque carácter*. El Spectrum tiene definidos tres tipos de caracteres: el conjunto de 96 caracteres *estándar,* los 16 *bloques gráficos* y el conjunto de 21 caracteres *definidos por el usuario.* Para tener acceso desde teclado a los dos últimos tipos debemos estar trabajando *en modo gráfico.* Estos caracteres son llevados a la pantalla por medio de la instrucción PRINT, de forma que deberíamos echar un vistazo a esta operación antes de seguir.

El comando PRINT nos permite colocar los caracteres en cualquiera de las 22 líneas superiores de la pantalla, desde la superior (0) hasta la inferior (21), y, dentro de una de ellas, en cualquiera de sus 32 posiciones posibles desde la izquierda (0) hasta la derecha (31). Estos son los bloques carácter y cada uno tiene 8 líneas de 8 *pixels.* Cada línea de ocho *pixels* se corresponde con un valor guardado en una posición de memoria. Cada *pixel,* por su parte, se corresponde con un dígito binario (bit) de un número binario de ocho dígitos (byte). Este número puede representarse en forma decimal, estando su valor dentro del margen 0-255. En la memoria existe una tabla de caracteres. Para un carácter dado, el comando PRINT, o bien encuentra los ocho valores en la tabla o bien los calcula; a continuación los copia en las zonas de memoria reservadas para presentar la información. Este proceso es el que produce la presentación del carácter en la pantalla.

El conjunto de caracteres estándar

La tabla de datos necesaria para la traducción del juego de caracteres estándar está almacenada en ROM, la memoria permanente de sólo lectura *(**R**ead **O**nly **M**emory).* Hay ocho datos para cada uno de los 96 caracteres, por tanto la tabla

consiste en 768 (96*8) posiciones de memoria consecutivas, empezando en la 15616. Cada carácter tiene un número de código único (véase el Manual de BASIC del Spectrum). La tabla contiene los datos correspondientes a cada uno de los caracteres, empezando con el espacio (código 32) y terminando con el símbolo de "copyright" (código 127). Cuando el comando PRINT necesita los ocho datos de un carácter, busca en la variable del sistema CHARS (véase, de nuevo, el Manual de BASIC), que contiene la dirección de una posición de memoria 256 (ocho veces el código, por ahorro de espacio) menos que el principio de la tabla de caracteres. Para encontrar la dirección del primer dato, multiplica el número de código del carácter por ocho y lo suma al valor de CHARS. Por último PRINT copia el dato a la zona de memoria de presentación y el carácter aparece en la pantalla.

Ejecute el siguiente programa, basado en el listado 1.2. Le enseñará el funcionamiento de este proceso mostrándole los cálculos realizados. En el capítulo 13 daremos una explicación detallada de cómo manejar un carácter en la zona de memoria de presentación, así que, de momento, continuaremos empleando el bloque (0, 0) solamente. La figura 5.1 es un ejemplo de lo que obtenemos del programa cuando le introducimos como dato el carácter "?".

Listado 5.1

```
  10 CLS : INPUT "caracter ?";A$
: IF A$="" THEN  GO TO 10
  20 LET A$=A$(1): PRINT AT 2,0;
"""";A$;"""" SELECCIONADO. CODIGO
(";A$;")=";CODE A$
  30 PRINT AT 4,0;"CALCULO DE PO
SICION DE LOS DATOS"
  40 LET CHARS=PEEK 23606+256*PE
EK 23607
  50 PRINT AT 6,3;"VARIABLE CHAR
S=";CHARS
  60 LET TABLE=CODE A$*8
  70 PRINT AT 7,3;"CODIGO A$ * 8
 =  ";TABLE
  80 LET START=CHARS+TABLE
  90 PRINT AT 8,17;"-------": PR
INT AT 9,2;"LOS DATOS EMPIEZAN E
N ";START
 100 LET CORNER=16384: PRINT AT
11,0;"POS. TABLA  VALOR  POS. PA
NTALLA"
 110 FOR I=0 TO 7
 120 LET VALUE=PEEK (START+I): L
ET MEMORY=CORNER+256*I
```

```
 130 PRINT AT I+13,2;START+I: PR
INT AT I+13,22;MEMORY
 140 PRINT AT I+13,13;VALUE
 150 POKE MEMORY,VALUE
 160 NEXT I
 170 INPUT "OTRA VEZ ? ";Y$: IF
Y$="n" THEN  STOP
 180 GO TO 10
```

```
?
"?" SELECCIONADO. CODIGO(?)=63
CALCULO DE POSICION DE LOS DATOS
    VARIABLE CHARS=15360
    CODIGO A$ * 8  =  504
                     -------
  LOS DATOS EMPIEZAN EN 15864
POS. TABLA   VALOR   POS. PANTALLA
   15864         0          16384
   15865         60         16640
   15866         66         16896
   15867         4          17152
   15868         8          17408
   15869         0          17664
   15870         8          17920
   15871         0          18176
```

Figura 5.1

Ejercicio 5.1

Reforme el listado 5.1 de manera que le permita decidir si pasar un dato a la zona de presentación o no. Si alguno de los ocho valores es rechazado, teclee un nuevo valor. Experimente cambiando uno o dos valores de un carácter.

El modo gráfico

La dirección contenida en CHARS es donde se podría esperar que estuviera el primer dato correspondiente al carácter de código 0; sin embargo, los caracteres cuyo código está entre el 0 y el 31, ambos inclusive, son o bien caracteres de control, o no se usan. Como los caracteres de control no tienen representación gráfica, no existen datos para ellos en la tabla. A pesar de que CHARS indica su di-

rección, los 256 (32*8) bytes que les corresponden no se emplean, y están a nuestra disposición para otros usos. Más aún, si seleccionamos un carácter en modo gráfico para el listado 5.1, o bien aparecerá como blanco, o bien obtendremos unos datos totalmente espúreos. Esto ocurre porque la tabla sólo contiene los datos correspondientes a los caracteres con códigos desde el 32 al 127 y los caracteres gráficos tienen un código mayor que 127; por tanto el programa está buscando en un sitio equivocado, ¡más allá de la tabla de datos! La tabla en la ROM viene seguida por el comienzo de la RAM, memoria de acceso aleatorio de lectura y escritura *(**R**andom **A**ccess **M**emory)*, que se emplea como almacén temporal, y así el programa está mirando la zona de *display file*, que ocupa los primeros 6K de la RAM.

Gráficos de bloques

Los datos necesarios para representar el conjunto de los gráficos de bloques no están guardados en la ROM, sino que se calculan a partir su código (CODE). Cada carácter gráfico tiene cuatro cuadrantes que se representan, como sí o no, por cada uno de los cuatro últimos bits del número de código. En un carácter gráfico, de bloques, las primeras cuatro líneas de ocho *pixels* son idénticas, y lo mismo ocurre con las cuatro segundas.

Suponiendo que sabemos que un carácter pertenece al conjunto de los gráficos de bloques, podemos generar un par de datos y usarlos cuatro veces cada uno para crear el bloque. En la figura 5.2 tenemos un ejemplo de esto.

El conjunto de gráficos de bloques puede emplearse para realizar gráficos de resolución media con los 64 × 44 cuadrantes. Basándonos en esta idea podemos ela-

```
     10001101  =  CODIGO ("▟")

     1000XXXX  =  CARACTER GRAFICO
                       DE BLOQUES.

     XXXX1101  =  CUATRO ESQUINAS
                       DEL CARACTER.
1111 1111  0000 1111

00001111  =  VALOR DE LAS PRIMERAS
                         4 LINEAS

11111111  =  VALOR DE LAS ULTIMAS
                         4 LINEAS
     00001111
     00001111
     00001111
     00001111
     11111111     =
     11111111
     11111111
     11111111
```

Figura 5.2

borar imágenes en dos colores bastante complicadas con gran rapidez. La figura 5.3 se dibujó por medio del programa dado en el listado 5.2. El programa dibuja una serie de círculos concéntricos pequeños en la subrutina "pixels gigantes". Esta surutina nos pregunta por el bloque de carácter que estará en la posición, a lo largo de cinco bloques hacia abajo y ocho a la derecha, guarda los datos de la zona del *display file* en la matriz S. Para cada par de líneas, toma ocho datos y los convierte en una tira de números BINarios (véase la línea 1130 y siguientes). Estas tiras emplean para construir la representación BINaria del código de un carácter gráfico cogiendo un par de bits de cada tira y colocándole delante el código de carácter gráfico (línea 1230).

Listado 5.2

```
 100 REM programa principal
 110 LET big pixels=1000
 120 FOR I=1 TO 18 STEP 3: CIRCL
E 32,156,I: NEXT I
 130 GO SUB big pixels
 140 STOP
1000 REM pixels gigantes
1010 DIM P(8): DIM S(40,8)
1020 FOR I=1 TO 8: READ P(I): NE
XT I: DATA 128,64,32,16,8,4,2,1
1030 DEF FN S(R,C)=16384+INT (R/
8)*2048+(R-INT (R/8)*8)*32+C
1040 INPUT "EL BLOQUE IZQ. SUP.
ESTA EN      FIL,COL ";ROW;" , ";
COL
1049 REM Almacena los datos del
fichero de pantalla en la matriz
 S en bloques 5*8
1050 FOR I=0 TO 4
1060 FOR J=0 TO 7
1070 LET P=FN S(ROW+I,COL)
1080 FOR K=0 TO 7
1090 LET S(I*8+J+1,K+1)=PEEK (P+
256*J+K)
1100 NEXT K: NEXT J: NEXT I
1109 REM Fija la pantalla a 20*3
2 que es 4 veces el area a expan
dir
1110 BORDER 1: PAPER 7: INK 0: C
LS
```

```
1120 PRINT AT 21,0; PAPER 1;,,:
PRINT AT 0,0; PAPER 1;,,
1129 REM Toma dos lineas de elem
entos de pantalla a la vez para
convertirlas en un cuarto de blo
que
1130 FOR I=1 TO 39 STEP 2
1140 LET A=I: LET B=I+1
1150 FOR J=1 TO 8
1160 LET A$="00000000": LET B$=A
$
1170 LET T=S(A,J): LET U=S(B,J)
1179 REM Calcula las formas bina
rias de los elementos de pantall
a para el bloque jesimo
1180 FOR K=1 TO 8
1190 IF T>=P(K) THEN  LET T=T-P(
K): LET A$(K)="1"
1200 IF U>=P(K) THEN  LET U=U-P(
K): LET B$(K)="1"
1210 NEXT K
1219 REM Toma dos elementos de p
antalla de cada par de bytes par
a hacer 4 cuartos
1220 FOR K=1 TO 4: LET D=2*K: LE
T C=D-1
1230 LET C$="BIN 1000"+B$(C TO D
)+A$(C TO D)
1239 REM Convierte la forma bina
ria de un cuarto de bloque a car
acter
1240 LET C=VAL C$: PRINT CHR$ C;
1250 NEXT K: NEXT J: NEXT I
1260 RETURN
```

Los caracteres gráficos generados por el programa anterior se representan en la pantalla y forman una imagen exacta de lo que hubiera en el rectángulo especificado de 5 × 8 bloques. Esta imagen ha sufrido un aumento de escala, en ambas direcciones, por un factor 4, de forma que ahora cubre 20 × 32 bloques y cada *pixel* original es representado ahora por 4 *pixels.*

Con la facilidad COPY y la impresora ZX, se pueden conseguir listados con una anchura y longitud cuatro veces mayor que la normal; sólo hay que emplear la subrutina “pixels gigantes” 16 veces y pegar los trozos correspondientes del listado.

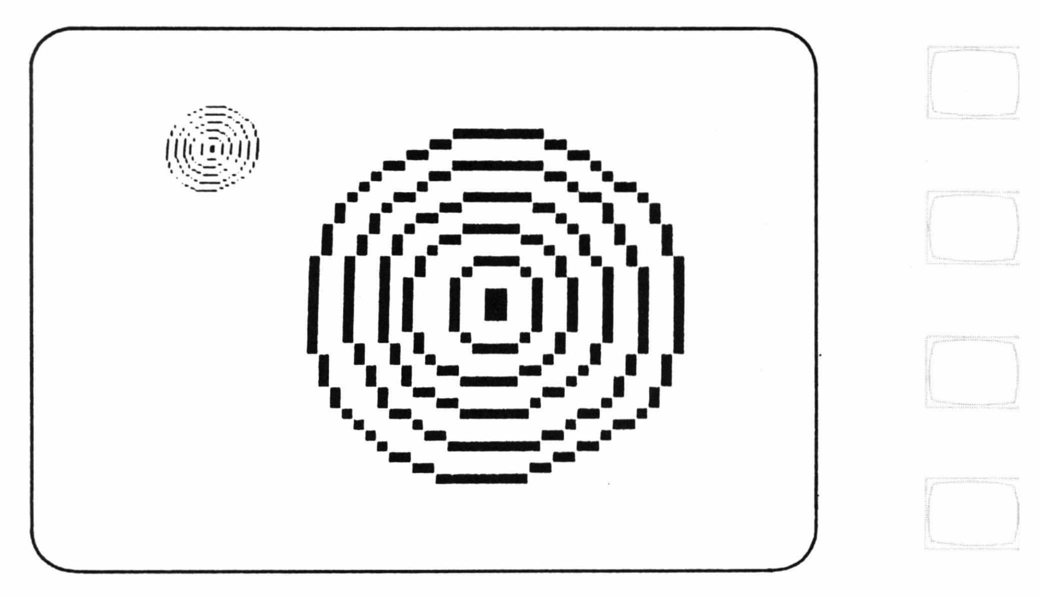


Figura 5.3

Ejercicio 5.2

Escriba una subrutina que produzca un listado de ella misma COPYando cada trozo de su listado de 5 × 8 como "pixels gigantes" en la impresora.

Gráficos definidos por el usuario

El tercer conjunto de caracteres es el conjunto de gráficos definidos por el usuario (códigos desde el 144 al 164). Existe otra tabla de datos para este juego de caracteres en las últimas 168 posiciones de memoria. Cuando encendemos la máquina, se copian en esta tabla los 21 caracteres desde la "A" hasta la "U". Los datos de esta tabla pueden cambiarse a voluntad, de forma que puede obtenerse desde el teclado cualquier carácter. Existe una función (USR), que nos da la dirección del primer dato del carácter correspondiente al que le mandamos como parámetro ("A" hasta "U"). La dirección del principio de esta tabla está guradada en la variable del sistema UDG (vea el Manual de BASIC del Spectrum). El código del caracter gráfico menos 144 y multiplicado por 8 nos da la posición relativa del primero de los ocho datos de dicho carácter dentro de la tabla. La función USR realiza estos cálculos y suma el resultado a UDG para obtener la dirección absoluta donde empieza el carácter. UDG puede variar y tener menos caracteres, por ejemplo sólo desde la A a la H, dejando más espacio a los programas; de todas formas, esto no suele ser necesario a menos que sólo disponga de 16K de RAM. Para ilustrar el empleo de esta función con el fin de cambiar un carácter gráfico, teclee la siguiente línea

```
POKE USR "A",BIN 00100100
```

Ahora, si entramos en modo gráfico y tecleamos "A", veremos que dos *pixels* adicionales han aparecido como dos puntos sobre la "A". El programa del listado 5.3 le permite redefinir los ocho valores binarios que representan un carácter gráfico. El programa simula el modo gráfico, permitiéndole representar caracteres gráficos definidos por usuario en la pantalla.

Listado 5.3

```
  10 INPUT "CARACTER A REDEFINIR
",U$: IF U$="" THEN  GO TO 10
  20 IF U$<"A" OR U$>"U" THEN  G
O TO 10
  30 LET MEMORY=USR U$: CLS
  40 FOR I=0 TO 7
  50 INPUT ("VALOR "+STR$ I+" =B
IN "); LINE B$
  60 LET VALUE=VAL ("BIN "+B$)
  70 POKE MEMORY+I,VALUE
  80 IF LEN B$<8 THEN  LET B$="0
"+B$: GO TO 80
  90 PRINT B$: NEXT I
 100 PRINT AT 10,0;"TECLEA CARAC
TERES O ENTER"
 109 REM Simula el modo GRAFICOS
 para las letras A a U
 110 LET R=12: LET C=0
 120 PRINT AT R,C; FLASH 1;"G"
 130 IF INKEY$<>"" THEN  GO TO 1
30
 140 LET A$=INKEY$: IF A$="" THE
N  GO TO 140
 149 REM Si se aprieta ENTER se
recomienza el programa
 150 IF A$=CHR$ 13 THEN  GO TO 1
0
 160 IF A$<"a" OR A$>"u" THEN  G
O TO 130
 169 REM Convierte el caracter a
l G.D.U. equivalente y lo imprim
e
 170 LET A$=CHR$ (CODE A$+47)
 180 PRINT AT R,C;A$
 189 REM Mueve los punteros apar
entes del cursor
```

```
 190 LET C=C+1: IF C=32 THEN  LE
T C=0: LET R=R+1: IF R=22 THEN
LET R=12
 200 PRINT AT R,C; FLASH 1;"G"
 210 GO TO 130
```

Ejercicio 5.3

Emplee el listado 5.3 para construir caracteres gráficos de bloques consistentes en dameros en lugar de zonas completamente coloreadas. Experimente con estos caracteres con varios colores en PAPER e INK para conseguir nuevos colores; por ejemplo, rojo y amarillo dan naranja.

Los caracteres definidos por el usuario puede incorporarlos directamente a sus programas para aumentar la velocidad de dibujo. El carácter equivalente podría ser construido por medio de una serie de comandos PLOT y DRAW, pero este sistema es, en la mayoría de los casos, mucho más lento. Por ejemplo, por medio de caracteres podemos construir modelos geométricos en la pantalla mucho más rápidamente que si los trazamos con PLOT y DRAW. El listado 5.4 nos muestra cómo puede hacerse esto por medio de los caracteres redefinidos "A" a "D" (los caracteres subrayados en el listado indican caracteres en modo gráfico).

Listado 5.4

```
 200 REM programa principal
 209 REM Llena la pantalla con d
iagonales de los 4 primeros cara
cteres graficos
 210 OVER 0
 220 INPUT "TINTA ";I: INK I
 230 INPUT "PAPEL ";I: PAPER I
 240 LET A$="    ": LET B$="
"
 250 LET C$="    ": LET D$="
"
 260 FOR J=0 TO 7
 270 FOR I=0 TO 4
 280 PRINT AT I*4,J*4;A$
 290 PRINT AT I*4+1,J*4;B$
 300 PRINT AT I*4+2,J*4;C$
 310 PRINT AT I*4+3,J*4;D$
 320 NEXT I
 330 PRINT AT I*4,J*4;A$
```

```
340 PRINT AT I*4+1,J*4;B$
350 NEXT J
360 STOP
```

Juegos de caracteres alternativos

Hasta ahora hemos empleado 37 caracteres gráficos para realizar los dibujos; a partir de ahora podremos emplear 96 caracteres adicionales por cada nuevo juego de ellos que definamos. Para emplear un nuevo conjunto de caracteres alternativo, todo lo que tenemos que hacer es cambiar el valor de CHARS de forma que apunte hacia otra dirección en la memoria RAM (justamente donde hemos definido anteriormente nuestro nuevo juego de caracteres) en lugar de hacia la tabla estándar. Como habrá podido apreciar, el esfuerzo necesario para la construcción de un carácter es lo suficientemente grande para suponer que la elaboración de todo un nuevo conjunto de caracteres sea una ardua tarea. Afortunadamente disponemos de un ordenador, así que dejémoslo que realice él el trabajo sucio. El listado 5.5 es un programa para generar y retocar caracteres, y en su diseño se tuvo en mente la idea de utilizar dichos caracteres en el desarrollo de programas gráficos.

Listado 5.5

```
   1 CLEAR 62294: INK 0: PAPER 7
: BORDER 7: OVER 1: LET N=0: REM
 Para ordenadores de 16K usar CL
EAR 255+ el valor de S(I), donde
 I es el indice del menor juego
de caracteres
   2 DIM Z$(8,8): DIM T$(8,8): D
IM S(6): DIM P(8): DIM B(8): DIM
 G(7): REM Numeros de linea del
GOTO para cada opcion del menu
   3 FOR I=1 TO 7: READ G(I): NE
XT I: DATA 61,67,71,100,106,112,
119: REM Potencias de 2 para las
 conversiones binarias
   4 FOR I=1 TO 8: READ P(I): NE
XT I: DATA 128,64,32,16,8,4,2,1:
 REM Valores a almacenar en la v
ariable del sistema CHARS para c
ada juego de caracteres. Para 16
K deben ser 15360,29271,30039,30
807,31575,32080.
```

```
   5 FOR I=1 TO 6: READ S(I): NE
XT I: DATA 15360,62039,62807,635
75,64343,64848
   6 LET S=1: GO SUB 17: REM Pon
e el juego de caracteres 1 e imp
rime el menu
   7 CLS : PRINT AT 2,1;"***GENE
RADOR DE CARACTERES***"
   8 PRINT AT 5,6; PAPER 5;"1";
PAPER 7;" ... IMPRIMIR JUEGOS"
   9 PRINT AT 7,6; PAPER 5;"2";
PAPER 7;" ... IMPRIMIR UN JUEGO"
  10 PRINT AT 9,6; PAPER 5;"3";
PAPER 7;" ... EDITAR CARACTER"
  11 PRINT AT 11,6; PAPER 5;"4";
 PAPER 7;" ... COPIA UN JUEGO EN
 OTRO"
  12 PRINT AT 13,6; PAPER 5;"5";
 PAPER 7;" ... GRABAR CARACTERES
"
  13 PRINT AT 15,6; PAPER 5;"6";
 PAPER 7;" ... CARGAR CARACTERES
"
  14 PRINT AT 17,6; PAPER 5;"7";
 PAPER 7;" ... EJECUTAR TU PROGR
AMA": REM Espera por una opcion
valida
  15 INPUT "OPCION: ";OP: IF OP<
1 OR OP>7 THEN  GO TO 15  : REM
Salta a la rutina apropiada
  16 GO TO G(OP): REM Las subrut
inas estan al principio para una
 mayor eficiencia.: REM Rutina p
ara cambiar caracteres alterando
 CHARS
  17 LET HI=INT (S(S)/256): LET
LO=S(S)-HI*256
  18 POKE 23606,LO: POKE 23607,H
I: RETURN : REM Rutina de espera
 antes de limpiar la pantalla y
volver al menu
  19 LET S=1: GO SUB 17 : INPUT
"ENTER PARA EL MENU"; LINE A$: R
ETURN : REM Limpia la matriz de
caracteres binarios
```

```
  20 FOR I=1 TO 8: LET Z$(I)="00
000000": NEXT I: RETURN : REM Le
e un numero de juego de caracter
es
  21 INPUT "NO. DE JUEGO ? ";S:
IF S<1 OR S>6 THEN  GO TO 21
  22 LET A=32: LET B=127: IF S=6
 THEN  LET A=65: LET B=85
  23 RETURN : REM Lee un caracte
r valido del juego escogido
  24 INPUT "CARACTER ?";C$: LET
C=CODE C$: IF C<A OR C>B THEN  G
O TO 24
  25 RETURN : REM Dibuja una rej
illa 8x8 en la posicion horizont
al alterada por N
  26 LET NN=64+N: FOR I=0 TO 8
  27 PLOT I*8+NN,64: DRAW 0,64
  28 PLOT NN,I*8+64: DRAW 64,0
  29 NEXT I: RETURN : REM Produc
e el equivalente grafico de la m
atriz de literales binarios en l
a rejilla
  30 FOR I=1 TO 8: FOR J=1 TO 8
  31 LET P=7: IF N=0 AND Z$(I,J)
="1" OR N=96 AND T$(I,J)="1" THE
N  LET P=4
  32 GO SUB 48 : NEXT J: NEXT I:
 RETURN : REM Imprime el cursor
en un cuadrado de la rejilla
  33 PLOT X,Y-2: DRAW 0,4
  34 PLOT X-2,Y: DRAW 4,0
  35 RETURN : REM Amplia una esq
uina del caracter en la matriz t
emporal T$ y la salva
  36 INPUT "ESQUINA ? ";C: IF C<
1 OR C>4 THEN  GO TO 36
  37 FOR I=1 TO 8: LET T$(I)="00
000000": NEXT I: GO TO 36+C*2
  38 FOR I=1 TO 4: FOR J=1 TO 4:
 IF Z$(I,J)="1" THEN  LET TI=2*I
: LET TJ=2*J: GO SUB 46
  39 NEXT J: NEXT I: RETURN
  40 FOR I=1 TO 4: FOR J=5 TO 8:
 IF Z$(I,J)="1" THEN  LET TI=2*I
```

```
: LET TJ=2*(J-4): GO SUB 46
  41 NEXT J: NEXT I: RETURN
  42 FOR I=5 TO 8: FOR J=1 TO 4:
 IF Z$(I,J)="1" THEN  LET TI=2*(
I-4): LET TJ=2*J: GO SUB 46
  43 NEXT J: NEXT I: RETURN
  44 FOR I=5 TO 8: FOR J=5 TO 8:
 IF Z$(I,J)="1" THEN  LET TI=2*(
I-4): LET TJ=2*(J-4): GO SUB 46
  45 NEXT J: NEXT I: RETURN : RE
M : REM Subrutina para convertir
 un elemento de pantalla en 4 en
 una matriz ampliada
  46 LET T$(TI,TJ)="1": LET T$(T
I-1,TJ)="1": LET T$(TI,TJ-1)="1"
: LET T$(TI-1,TJ-1)="1": RETURN
: REM Imprime las etiquetas de i
dentificacion en las esquinas de
 la rejilla
  47 PRINT AT 5,7;"1": PRINT AT
5,16;"2": PRINT AT 14,7;"3": PRI
NT AT 14,16;"4": RETURN : REM Su
brutina para imprimir un bloque
de color en un cuadrado de la re
jilla
  48 PRINT AT I+5,J+7+N/8; PAPER
 P;" ": RETURN : REM Imprime el
caracter ampliado en otra rejill
a y lo salva
  49 LET N=96: GO SUB 26 : GO SU
B 30
  50 GO SUB 21 : GO SUB 24 : LET
 D=S(S)+C*8-1
  51 FOR I=1 TO 8: POKE D+I,VAL
("BIN "+T$(I)): NEXT I
  52 LET N=0: CLS : RETURN : REM
 Copia el simetrico de la matriz
 Z$ respecto al eje X en T$
  53 FOR I=1 TO 8: FOR J=1 TO 8
  54 LET T$(9-I,J)=Z$(I,J): NEXT
 J: NEXT I: GO SUB 59 : RETURN :
 REM Copia la matriz Z$ rotada e
n la T$
  55 FOR I=1 TO 8: FOR J=1 TO 8
  56 LET T$(9-J,I)=Z$(I,J): NEXT
```

```
J: NEXT I: GO SUB 59 : RETURN :
 REM Copia el simetrico de Z$ re
specto al eje Y en T$
  57 FOR I=1 TO 8: FOR J=1 TO 8
  58 LET T$(I,9-J)=Z$(I,J): NEXT
 J: NEXT I: GO SUB 59 : RETURN :
 REM Rutina para copiar la matri
z T$ de nuevo en la Z$
  59 FOR I=1 TO 8: FOR J=1 TO 8
  60 LET Z$(I,J)=T$(I,J): NEXT J
: NEXT I: RETURN : REM Comienzo
de las rutinas de manejo de cara
cteres: REM Imprime los 5 juegos
 de caracteres completos y los G
.D.U.
  61 CLS : FOR S=1 TO 5: GO SUB
17
  62 PRINT AT S*4-4,0;
  63 FOR C=32 TO 127: PRINT CHR$
 C;: NEXT C
  64 NEXT S: GO SUB 17 : PRINT A
T 20,0;
  65 FOR C=65 TO 85: PRINT CHR$
C;: NEXT C: REM Llama a la rutin
a de espera
  66 GO SUB 19 : GO TO 7 : REM I
mprime un juego y los caracteres
 que sustituye
  67 CLS : GO SUB 21 : LET SS=S
  68 FOR C=A TO B: PRINT  PAPER
6;CHR$ C;"=";: LET S=SS: GO SUB
17
  69 PRINT CHR$ C;: LET S=1: GO
SUB 17 : PRINT " ";: NEXT C: REM
 Vuelve al menu
  70 GO SUB 19 : GO TO 7 : REM O
pcion 3, edita un caracter limpi
ando la pantalla y Z$
  71 CLS : GO SUB 20 : REM Calcu
la la posicion de un caracter
  72 GO SUB 21: GO SUB 24: LET D
=S(S)+C*8-1: REM Toma 8 numeros
binarios de la tabla de datos
  73 FOR I=1 TO 8: LET B(I)=PEEK
 (D+I): NEXT I: REM Convierte el
```

```
numero en un literal binario de
 8 digitos
  74 FOR I=1 TO 8: LET T=B(I)
  75 FOR J=1 TO 8
  76 IF T>=P(J) THEN  LET Z$(I,J
)="1": LET T=T-P(J)
  77 NEXT J: NEXT I: REM Dibuja
una rejilla con los bloques verd
es representando un 1 binario
  78 GO SUB 26 : GO SUB 30 : REM
 Inicializa la posicion del curs
or y los valores de referencia d
e la rejilla
  79 LET X=68: LET Y=124: LET I=
1: LET J=1: REM Sobreimpresiona
el cursor
  80 GO SUB 33 : REM Espera hast
a apretar una tecla
  81 IF INKEY$<>"" THEN  GO TO 8
1
  82 IF INKEY$="" THEN  GO TO 82
  : REM Lee el comando y borra e
l cursor
  83 LET A$=INKEY$: BEEP 0.05,30
: GO SUB 33 : REM Comprueba si s
e estan apretando las teclas del
 cursor
  84 IF (A$="5" OR A$=CHR$ 8) AN
D J>1 THEN  LET X=X-8: LET J=J-1
  85 IF (A$="6" OR A$=CHR$ 10) A
ND I<8 THEN  LET Y=Y-8: LET I=I+
1
  86 IF (A$="7" OR A$=CHR$ 11) A
ND I>1 THEN  LET Y=Y+8: LET I=I-
1
  87 IF (A$="8" OR A$=CHR$ 9) AN
D J<8 THEN  LET X=X+8: LET J=J+1
: REM Comandos especiales: REM R
ellenado de un cuadrado
  88 IF A$="P" OR A$="p" THEN  L
ET P=4: LET Z$(I,J)="1": GO SUB
48 : REM Borrado de un cuadrado
  89 IF A$="O" OR A$="o" THEN  L
ET P=7: LET Z$(I,J)="0": GO SUB
48 : REM Rota el caracter 90 gra
```

```
dos en sentido antihorario
  90 IF A$="R" OR A$="r" THEN  G
O SUB 55: GO SUB 30: GO TO 79: R
EM Simetrica respecto a X
  91 IF A$="X" OR A$="x" THEN  G
O SUB 53: GO SUB 30: GO TO 79: R
EM Simetrica respecto a Y
  92 IF A$="Y" OR A$="y" THEN  G
O SUB 57: GO SUB 30: GO TO 79: R
EM Mezcla un caracter con el act
ual
  93 IF A$="M" OR A$="m" THEN  G
O SUB 26: GO TO 72  : REM Amplia
 al doble una esquina del caract
er
  94 IF A$="D" OR A$="d" THEN  G
O SUB 47 : GO SUB 36 : GO SUB 49
 : GO TO 78  : REM Vuelve al pri
ncipio o salva el caracter
  95 IF A$<>"S" AND A$<>"s" THEN
  GO TO 80  : REM Sustituye el c
aracter en la tabla de datos usa
ndo la funcion BIN
  96 GO SUB 21 : GO SUB 24
  97 LET D=S(S)+C*8-1
  98 FOR I=1 TO 8: POKE D+I,VAL
("BIN "+Z$(I)): NEXT I: REM vuel
ve al menu
  99 GO SUB 19: GO TO 7: REM Opc
ion 4, copia un juego en otro
 100 INPUT "DEL JUEGO ";A;" AL J
UEGO ";B: REM Comprueba si los j
uegos son validos
 101 IF A<1 OR B<1 OR A>5 OR B>5
 THEN  GO TO 100
 102 IF A=B THEN  GO TO 105 : RE
M Copia los datos de una tabla a
 otra
 103 PRINT AT 21,10;"ESPERE": FO
R K=256 TO 1024
 104 POKE (S(B)+K),PEEK (S(A)+K)
 : NEXT K: REM Vuelve al menu
 105 GO SUB 19 : GO TO 7 : REM O
pcion 5, graba un juego de carac
teres en cinta
```

```
 106 CLS : PRINT AT 2,6;"GRABAND
O CARACTERES": INPUT "NOMBRE ? "
;N$: IF N$="" THEN  GO TO 7
 107 PRINT AT 4,10;N$: INPUT "JU
EGO ? ";S: IF (S<2 OR S>6) AND S
<>11 THEN  GO TO 107 : REM Si te
cleas 5+6 graba los juegos 5 y 6
 108 IF S=11 THEN  SAVE N$ CODE
(S(5)+256),168+768: GO TO 111 :
REM El juego 6 es menor que los
otros
 109 IF S=6 THEN  SAVE N$ CODE (
S(6)+512),168: GO TO 111
 110 SAVE N$ CODE (S(S)+256),768
: REM Vuelve al menu
 111 GO SUB 19 : GO TO 7 : REM O
pcion 6, carga un juego de carac
teres de cinta
 112 CLS : PRINT AT 2,6;"CARGAND
O CARACTERES": INPUT "NOMBRE ? "
;N$: IF N$="" THEN  GO TO 7
 113 PRINT AT 4,10;N$: INPUT "JU
EGO ? ";S: IF (S<2 OR S>6) AND S
<>11 THEN  GO TO 113
 114 PRINT AT 21,10;"Arranca el
casete.": REM Si tecleas 5+6 car
ga los juegos 5 y 6
 115 IF S=11 THEN  LOAD N$ CODE
(S(5)+256),168+768: GO TO 118: R
EM Carga el juego definido por e
l usuario
 116 IF S=6 THEN  LOAD N$ CODE (
S(6)+512),168: GO TO 118
 117 LOAD N$ CODE (S(S)+256),768
: REM Vuelve al menu
 118 GO SUB 19 : GO TO 7
```

Las subrutinas GENERADORAS DE CARACTERES han sido pensadas para que realicen todos los trabajos desagradables de preparar y usar los caracteres definidos: le permiten editar y manipular los caracteres, emplear juegos de caracteres alternativos, guardar y leer de cinta dichos conjuntos de caracteres y probar inmediatamente sus programas. Las subrutinas han sido diseñadas para el Spectrum de 48K y su localización será en la parte inferior de la memoria (líneas 1 a 118); a continuación irá su programa (con números de línea superiores al 118), y por último se añadirán los programas de renumeración y borrado (están en el capítulo 13, no los

busque por aquí) que comienzan en la línea 9900. Si posee un Spectrum de 16K, entonces este programa debería ejecutarse independientemente de otras subrutinas (vea el Apéndice A).

En primer lugar, el programa ofrece una serie de siete opciones que describiremos a continuación:

1) La primera opción es IMPRIMIR JUEGOS. Esta opción imprimirá el juego de caracteres normal (juego 1), seguido de los cuatro juegos alternativos (2 a 5) y por último del conjunto gráfico definido por el usuario (6). Los últimos caracteres del juego 5 contendrán datos espúreos, restos de la pila del GO SUB, que ha sido cambiada de sitio por razones de seguridad.

2) La segunda es IMPRIMIR UN JUEGO. Estos están identificados por los números del 1 al 6: el conjunto número 1 es el juego de caracteres estándar y no puede cambiarse; del 2 al 5 son conjuntos alternativos y pueden reemplazar temporalmente al estándar; el 6 es el juego de caracteres gráficos definidos por el usuario y siempre estará a nuestra disposición. Al seleccionar uno de estos números, la pantalla nos mostrará cada carácter del juego estándar seguido del signo igual (=), ambos sobre fondo amarillo, seguidos del carácter equivalente el juego alternativo. En el caso del juego 6, sólo se presentarán los caracteres desde "A" hasta "U", y mostrará los que se obtienen al utilizarlos en modo gráfico.

3) La opción 3 es el editor. Esta es la opción más compleja y conlleva el uso de varios comandos que se ejecutan tecleando su inicial. En primer lugar se le pregunta por el conjunto de caracteres que desea modificar y, dentro de ellos, por el carácter en particular. Si quiere empezar con una rejilla negra de 8 × 8 *pixels,* elija el juego 1 y el carácter espacio. (Para seleccionar las comillas es necesario teclearlas dos veces (vea el Manual de BASIC del Spectrum).)

Se encontrará ahora en modo de edición y el carácter elegido es presentado en forma de rejilla de 8 × 8 bloques de color verde. La cruz de la esquina superior izquierda es el cursor de edición. La posición de este cursor se controla por medio de las teclas usuales ("5", "6", "7" yu "8") no siendo necesario tener oprimida la tecla de mayúsculas.

Los dos primeros comandos son PLOT u OFF, que cambian un cuadro en la rejilla del carácter (PLOT lo dibuja y OFF lo borra). Coloque el cursor sobre el cuadro en cuestión y teclee "P" u "O".

"P" vuelve el cuadro de color verde, equivalente a poner un uno binario o a dibujar el *pixel* correspondiente, y "O" lo borra quedando en blanco.

Los tres comandos siguientes realizan transformaciones similares a las que vimos en el capítulo anterior. ROTACION gira el carácter 90 grados, en sentido contrario a las agujas del reloj, alrededor de su centro. X-EJE refleja el carácter a lo largo del eje horizontal, e Y-EJE lo refleja según el vertical. Estos comandos pueden emplearse para crear textos en cualquier orientación.

Por último, tenemos MEZCLA, SAVE y DOBLE. MEZCLA permite sobreimponer otro carácter sobre el que estamos editando. Es de gran utilidad para la creación de juegos de caracteres propios de un idioma; por ejemplo, la letra "ñ" y las vocales

acentuadas en el caso del español, o de la "c" con cedilla ("ç") en el caso del catalán, etc. El comando SAVE pregunta en qué juego de caracteres queremos guardar el recién creado y a cuál de los estándares queremos asociarlo. Si se especifica el juego 1, el carácter se perderá, ya que se encuentra en la memoria ROM del Spectrum: este proceso nos permite desembarazarnos de algún carácter creado por equivocación y que contiene suficiente número de errores para no querer cambiarlos uno a uno. DOBLE es un comando muy útil que nos permite coger un cuadrante del carácter que estamos editando y aumentarlo hasta que ocupe toda la rejilla, y guardarlo (mediante SAVE) como un carácter. Esta opción no afecta al carácter en edición, de forma que pueden guardarse las cuatro esquinas en cuatro caracteres distintos sin tener que iniciar la edición para cada una de ellas. Esta característica puede usarse para crear caracteres de tamaño doble, cuádruple e incluso mayor con gran facilidad.

4) COPIA UN JUEGO EN OTRO es la cuarta opción y puede emplearse para hacer copias de un juego de caracteres completo para manipularlas a continuación, o simplemente para cambiar un juego de caracteres a otra posición de memoria. Los juegos 2 a 5 están almacenados al final de la memoria, justo antes de la tabla de caracteres gráficos definidos por el usuario, con el conjunto 5 en último lugar. Esta área de memoria está reservada y protegida del BASIC por medio del comando CLEAR 62294 en la versión de 48K (para más detalles, consulte el Manual de BASIC del Spectrum). 62294 es la posición anterior al comienzo de la tabla para el juego 2. Para proteger las zonas correspondientes a los conjuntos I y siguientes, tenemos que evaluar S(I) + 255 y colocar este número en el comando CLEAR al principio del programa. S(I) es el valor asignado a la variable CHARS que permite a la instrucción PRINT el acceso al juego I. Estos valores son leídos (mediante la sentencia READ) de los datos correspondientes a la línea 3 (sentencia DATA). Si dispone de una máquina con 16K, siga las instrucciones del Apéndice A.

5) y 6). La opción quinta nos permite guardar todo un juego de caracteres en cinta y la sexta recuperarlos. Si en respuesta a la pregunta "NO. DE JUEGO", tecleamos un número entre 1 y 6, entonces este juego será guardado o cargado. Si tecleamos, por ejemplo, 5 + 6, entonces los juegos 5 y 6 se guardan o cargan como una unidad conjunta.

Para que otros programas puedan cargar y emplear los juegos alternativos de caracteres creados, tendremos que copiar las líneas 112 a 117 del GENERADOR DE CARACTERES" además de la subrutina (17 y 18), que permite saltar de un conjunto a otro. El vector S y sus datos de la línea 5 deberán incluirse también en nuestros programas.

7) La última opción es EJECUTAR TU PROGRAMA, que transfiere el control a la primera línea que siga al generador de caracteres; si no existe esta sentencia, el programa se parará.

Para que usted se familiarice con el "GENERADOR DE CARACTERES" siga las instrucciones que se dan a continuación.

Una los listados 5.4 y 5.5 (con el comando MERGE) en el Spectrum de 48K; o bien ejecútelos por separado en el de 16K. Cree un carácter en forma de borrón de tinta y guárdelo como carácter "**A**" en el juego 6. Edite ese carácter: gírelo y guárdelo como "**B**" del juego 6. Edite "**A**" del 6: use X-EJE y guárdelo como "**C**" del 6. Edite "**B**": use la reflexión Y-EJE y guárdelo como "**D**". Ahora utilice la opción 7 para ejecutar su programa y vea qué figura obtiene en la pantalla.

Ejercicio 5.4

Experimente con algunas de las simetrías posibles de caracteres y modelos colocando caracteres en la pantalla. Cambie el programa de forma que genere todas las combinaciones de INK y PAPER por medio de los dos bucles FOR ... NEXT anidados.

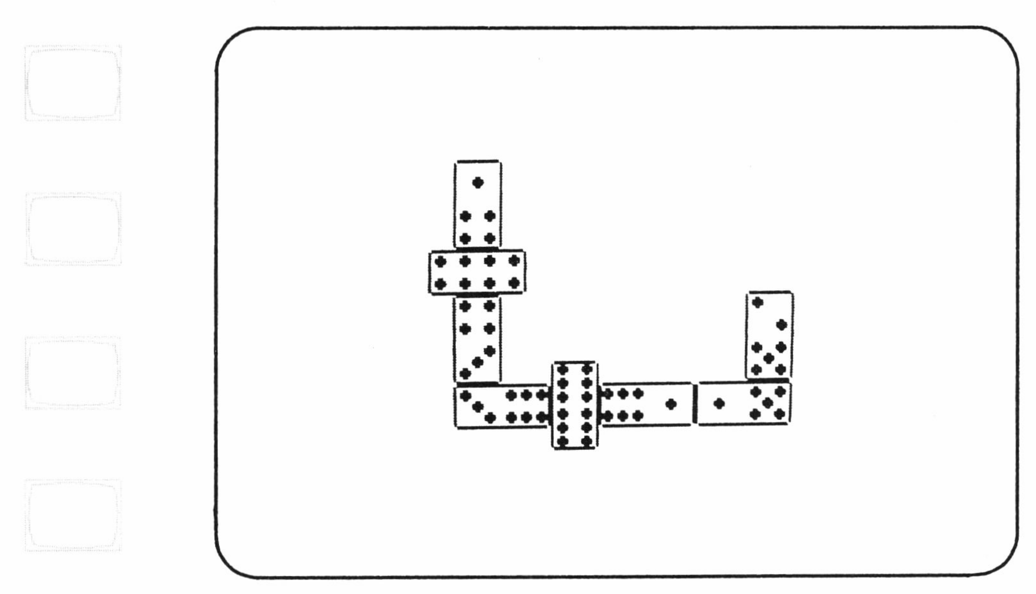


Figura 5.4

Ejercicio 5.5

Cree caracteres que representen fichas de dominó por medio del "GENERADOR DE CARACTERES" del listado 5.5 Esto le permitirá construir imágenes como la de la figura 5.4.

Ejercicio 5.6

Escriba una subrutina que copie un conjunto de caracteres en otro, pero que emplee alguna subrutina del editor del "GENERADOR DE CARACTERES" para efectuar alguna transformación antes de copiarlos. La figura 5.5 muestra un listado producido con un juego de caracteres que ha sido girado.

Figura 5.5

Consideremos ahora un programa completo que ha sido desarrollado empleando el "GENERADOR DE CARACTERES" y separado a continuación del sistema de desarrollo para que funcione en solitario. El programa "MASTER MIND" (listado 5.6), una de cuyas imágenes de cuando el juego se está desarrollando se presenta en la figura 5.6, se ha creado basándonos en un juego de caracteres alternativo. Este juego de caracteres se ha generado DOBL(E)ando las letras y dígitos necesarios del conjunto 1 y guardándolos en las esquinas superior derecha y superior izquierda de las letras mayúsculas del juego 5 y las esquinas inferiores en las letras minúsculas del mismo juego. Los dos tamaños de clavijas se guardaron en el conjunto de caracteres gráficos definidos por el usuario, que es independiente del juego principal; por tanto están siempre disponibles, ya estemos trabajando con el juego 1 o con el 5. Las letras de doble tamaño fueron editadas para redondear sus bordes y los conjuntos 5 y 6 (5 + 6) se guardaron en cinta bajo el nombre de "master".

Listado 5.6

```
   1 CLEAR 62294: INK 0: PAPER 7
: BORDER 7: REM Rutinas para el
uso de juegos de caracteres alte
rnos: REM Hacer las alteraciones
 del listado 5.5 para 16K
   2 DIM S(6)
   5 FOR I=1 TO 6: READ S(I): NE
XT I: DATA 15360,62039,62807,635
75,64343,64848
```

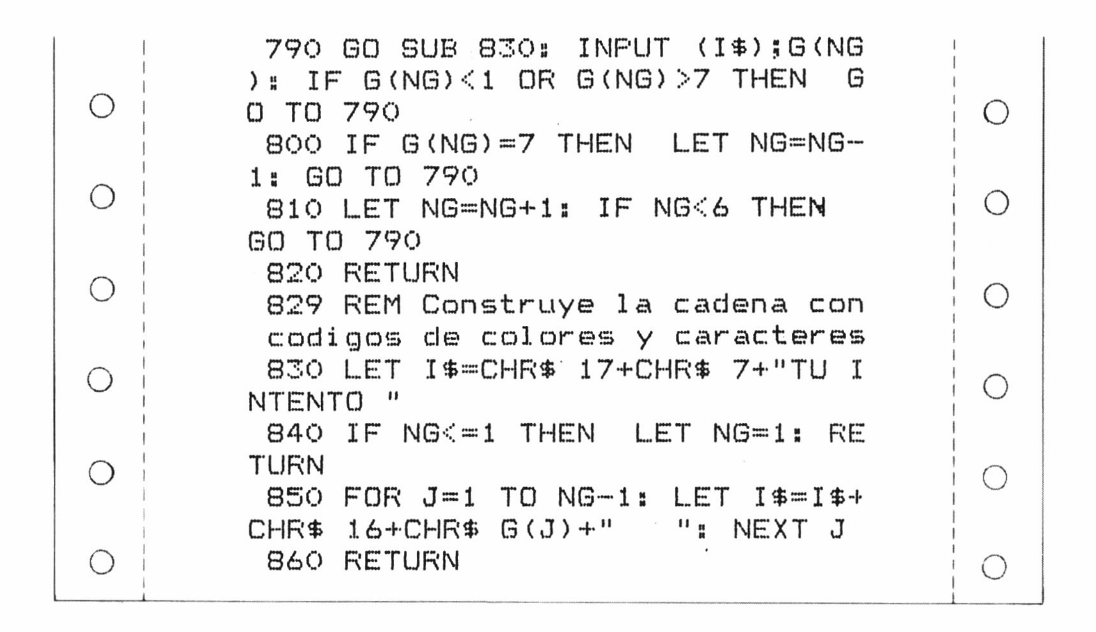
```
   6 LET S=1: GO SUB 17
   7 GO TO 200
  17 LET HI=INT (S(S)/256): LET
LO=S(S)-HI*256
  18 POKE 23606,LO: POKE 23607,H
I: RETURN
 112 CLS : PRINT AT 2,6;"CARGAND
O CARACTERES": LET N$="master"
 113 PRINT AT 4,10;N$: LET S=5+6
 114 PRINT AT 21,10;"Arranca el
casete.": PRINT AT 5,0;
 115 LOAD N$ CODE (S(5)+256),168
+768
 118 RETURN

 198 REM Carga los caracteres e
inicializa la puntuacion
 199 REM Matrices para guardar e
l codigo y el intento
 200 GO SUB 112: LET MYSCORE=0:
LET SCORE=0: DIM G(5): DIM T(4):
 DIM X(4)
 209 REM Rutina para dibujar la
pantalla
 210 RANDOMIZE : GO SUB 480
 219 REM Fija los colores de la
clave y va al bucle de intentos
 220 FOR I=1 TO 4: LET T(I)=INT
(RND*6)+1: NEXT I: LET GO=0
 229 REM Siguiente intento, si e
s el septimo pierdes
 230 LET GO=GO+1: IF GO=7 THEN
GO TO 390
 239 REM Rutina de entrada del i
ntento
 240 GO SUB 780
 249 REM Imprime el intento en e
l tablero (A es el grafico A)
 250 PAPER 7
 260 PRINT AT 1+GO*3,11; INK G(1
);"  "; INK G(2);"    "; INK G(3);
"    "; INK G(4);"    "
 269 REM Compara el intento con
una copia temporal del codigo
 270 LET NB=0: LET NW=0: FOR I=1
```

```
TO 4: LET X(I)=T(I): NEXT I
 279 REM Primero comprueba acier
tos exactos
 280 FOR I=1 TO 4: IF G(I)=X(I)
THEN  LET X(I)=0: LET NB=NB+1: L
ET G(I)=7
 290 NEXT I
 299 REM Comprueba colores acert
ados en posicion incorrecta
 300 FOR I=1 TO 4: FOR J=1 TO 4:
 IF G(I)=X(J) THEN  LET NW=NW+1:
 LET X(J)=0: LET G(I)=7
 310 NEXT J: NEXT I
 319 REM Fija la matriz X con el
 numero apropiado de blancos y n
egros
 320 FOR I=1 TO NB: LET X(I)=0:
NEXT I: FOR I=NB+1 TO NB+NW
 330 LET X(I)=7: NEXT I: IF NB+N
W<4 THEN  FOR I=NB+NW+1 TO 4: LE
T X(I)=4: NEXT I
 339 REM Dibuja los indicadores
blancos y negros en el tablero
 340 PAPER 4: PRINT AT GO*3+1,2;
 INK X(1);" "; INK X(2);" ";
 350 PRINT  INK X(3);" "; INK X(
4);" "
 359 REM Si tienes todo correcto
 ganas
 360 IF NB=4 THEN  GO TO 400
 369 REM Suma 1 a la puntuacion
del ordenador y pasa al siguient
e intento
 370 LET MYSCORE=MYSCORE+1: PRIN
T AT 10,26; PAPER 6; INK 0;MYSCO
RE
 380 GO TO 230
 389 REM Si pierdes el ordenador
 consigue 10 puntos mas
 390 LET MYSCORE=MYSCORE+10: GO
TO 410
 399 REM Si ganas consigues 10 p
untos
 400 LET SCORE=SCORE+10
 409 REM Construye el literal pa
```

```
ra imprimir en el tablero
 410 LET I$=CHR$ 17+CHR$ 7+"CLAV
E      "+CHR$ 16+CHR$ T(1)+"    "
 420 LET I$=I$+CHR$ 16+CHR$ T(2)
+"    "
 430 LET I$=I$+CHR$ 16+CHR$ T(3)
+"    "
 440 LET I$=I$+CHR$ 16+CHR$ T(4)
+" "+CHR$ 16+CHR$ 0
 450 LET I$=I$+CHR$ 6+"ENTER PAR
A CONTINUAR          "
 459 REM Muestra el codigo y esp
era hasta que se da ENTER para r
ecomenzar
 460 INPUT (I$); LINE B$
 470 GO TO 210
 479 REM Dibuja el tablero
 480 OVER 0: BORDER 6: PAPER 7:
CLS : INK 0
 489 REM Usa los caracteres alte
rnos para imprimir MASTER MIND a
mpliado
 490 LET S=5: GO SUB 17
 500 PRINT AT 0,0;" ABCDEFGHIJKL
  ABMNOPQR"
 510 PRINT AT 1,0;" abcdefghijkl
  abmnopqr"
 519 REM Imprime numeros ampliad
os
 520 PRINT AT 4,7;"ST": PRINT AT
 5,7;"st"
 530 PRINT AT 7,7;"UV": PRINT AT
 8,7;"uv"
 540 PRINT AT 10,7;"WX": PRINT A
T 11,7;"wx"
 550 PRINT AT 13,7;"YZ": PRINT A
T 14,7;"yz"
 560 PRINT AT 16,7;"[\": PRINT A
T 17,7;"{|"
 570 PRINT AT 19,7;"]^": PRINT A
T 20,7;"}~"
 579 REM Pinta los bordes del ta
blero en amarillo
 580 PAPER 4: FOR I=3 TO 20: PRI
NT AT I,1;"        ": NEXT I
```

```
 590 PAPER 6: FOR I=0 TO 21: PRI
NT AT I,24;"         ": NEXT I
 600 PAPER 7: FOR I=4 TO 19 STEP
 3
 609 REM Imprime espacios para l
os controles de aciertos
 610 PRINT AT I,11;"          ":
 NEXT I
 620 LET S=1: GO SUB 17: PAPER 2
 630 LET G$="      POR BRIAN JON
ES   Y    IAN  ANGELL 1982
 "
 640 FOR I=0 TO 8
 650 PRINT AT 12+I,24; PAPER 6;
INK 2;CHR$ 133; PAPER 2; INK 7;G
$(I*6+1 TO I*6+6);
 660 PRINT  INK 2; PAPER 6;CHR$
138: NEXT I
 669 REM Imprime las puntuacione
s
 670 PAPER 6: PRINT AT 2,26;"TUS
": PRINT AT 3,25;"PUNTOS"
 680 PRINT AT 5,26;SCORE
 690 PRINT AT 7,26;"MIS": PRINT
AT 8,25;"PUNTOS"
 700 PRINT AT 10,26;MYSCORE
 709 REM Dibuja lineas horizonta
les en el tablero
 710 FOR I=2 TO 20 STEP 3: LET Y
=168-8*I
 720 PLOT 7,Y: DRAW 176,0: PLOT
7,Y-1: DRAW 176,0: NEXT I
 729 REM Dibuja lineas verticale
s en el tablero
 730 PLOT 7,7: DRAW 0,145: PLOT
8,7: DRAW 0,145
 740 PLOT 183,7: DRAW 0,145: PLO
T 184,7: DRAW 0,145
 750 PLOT 55,7: DRAW 0,145: PLOT
 56,7: DRAW 0,145
 760 PLOT 71,7: DRAW 0,145: PLOT
 72,7: DRAW 0,145
 770 RETURN
 779 REM Controles de aciertos
 780 LET NG=1
```



```
 790 GO SUB 830: INPUT (I$);G(NG
): IF G(NG)<1 OR G(NG)>7 THEN G
O TO 790
 800 IF G(NG)=7 THEN  LET NG=NG-
1: GO TO 790
 810 LET NG=NG+1: IF NG<6 THEN
GO TO 790
 820 RETURN
 829 REM Construye la cadena con
 codigos de colores y caracteres
 830 LET I$=CHR$ 17+CHR$ 7+"TU I
NTENTO "
 840 IF NG<=1 THEN  LET NG=1: RE
TURN
 850 FOR J=1 TO NG-1: LET I$=I$+
CHR$ 16+CHR$ G(J)+"    ": NEXT J
 860 RETURN
```

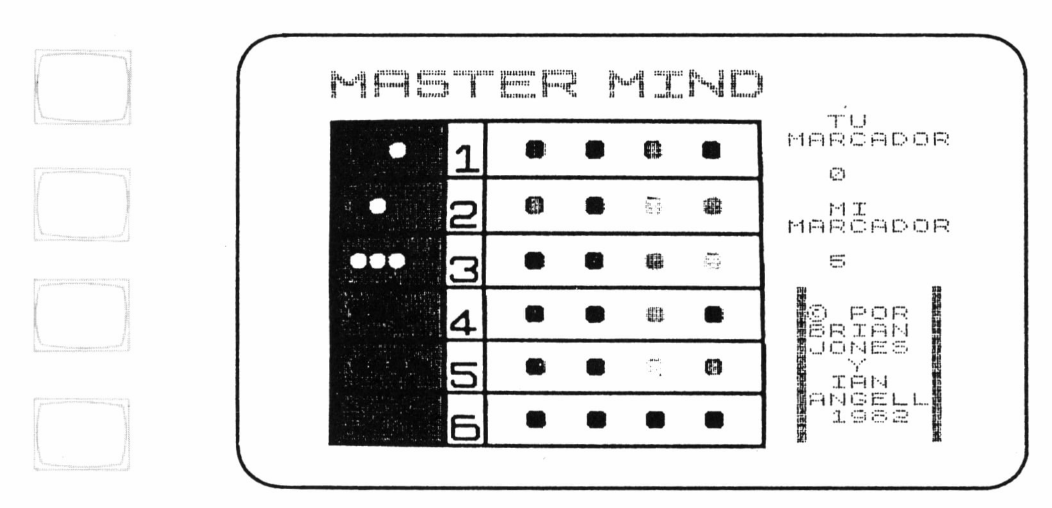


Figura 5.6

Las subrutinas copiadas del programa "GENERADOR DE CARACTERES" mantienen su numeración original de forma que puedan ser identificadas fácilmente y comparadas con el original. La imagen de la parte superior de la pantalla está construido por bloques sencillos a partir de los caracteres, colores y líneas de forma que el papel que juega cada bloque en la construcción es directo y encajable con gran facilidad. El programa emplea también la técnica de entrada dinámica de tiras de caracteres (que discutiremos en el capítulo 13) para crear la imagen de la parte inferior de la pantalla.

Ejercicio 5.7

Adecente un poco el programa "MÁSTER MIND", dándole una apariencia estructurada. Aumente su facilidad de lectura añadiéndole nombres de variable a las subrutinas. Adapte el programa "MASTER MIND" para que juegue contra usted.

Como final de este capítulo, damos un ejemplo de cómo conseguir imágenes de alta calidad por medio de una adecuada elección de caracteres gráficos. El pequeño programa siguiente emplea los caracteres de ajedrez de la cinta "ajedrez" para dibujar un tablero de ajedrez (evidentemente; véase la figura 5.7) y mover las piezas de acuerdo con las órdenes tecleadas. Por supuesto, usted puede crear sus propios caracteres para las fichas si éstos no le convencen.

Listado 5.7

```
   8 REM Poner el ordenador en m
ayusculas
   9 REM Inicializacion para usa
r los juegos 1 y 5. Para 16K usa
r CLEAR 31830; S5=31575
  10 CLEAR 64598: LET S5=64343:
LET S1=15360
  20 INPUT " CARGAR CARACTERES ?
 ";Y$: IF Y$="S" THEN  LOAD "aje
drez"CODE S5+256,768
 100 REM genera el tablero

 110 INK 0: PAPER 0: BORDER 0: C
LS
 120 PAPER 6
 129 REM Pinta de amarillo los b
ordes
 130 FOR I=1 TO 20: FOR J=1 TO 2
 140 PRINT AT 21-I,J+5;" ": PRIN
T AT I,J+23;" "
 150 PRINT AT J,I+5;" ": PRINT A
T J+18,26-I;" "
 160 NEXT J: NEXT I
 169 REM Imprime los cuadrados d
el tablero
 170 LET C=4
 180 FOR I=2 TO 8 STEP 2: FOR J=
2 TO 8 STEP 2
 190 PAPER C: PRINT AT J+1,I+6;"
  ": PRINT AT J+2,I+6;"  "
 200 PRINT AT 19-J,24-I;"  ": PR
INT AT 20-J,24-I;"  "
 210 LET C=7-C: PAPER C
```

```
 220 PRINT AT J+1,24-I;"  ": PRI
NT AT J+2,24-I;"  "
 230 PRINT AT 19-J,I+6;"  ": PRI
NT AT 20-J,I+6;"  "
 240 NEXT J: LET C=7-C: NEXT I
 249 REM Imprime letras y numero
s alrededor del tablero
 250 PAPER 6
 260 FOR I=1 TO 8: PRINT AT 20,6
+2*I;CHR$ (64+I): PRINT AT 2*I+2
,25;(9-I): NEXT I
 300 REM coloca las piezas
 310 DIM B(8,8)
 319 REM Las blancas tienen 10 s
umado al numero identificador de
 pieza
 320 FOR I=1 TO 8: LET B(2,I)=6:
 LET B(7,I)=16: NEXT I
 330 FOR I=1 TO 3: LET B(1,I)=4-
I: LET B(1,I+5)=I
 340 LET B(8,I)=14-I: LET B(8,I+
5)=I+10: NEXT I
 350 FOR I=4 TO 5: LET B(1,I)=9-
I: LET B(8,I)=19-I: NEXT I
 359 REM Fija los punteros de la
s rutinas
 360 LET move=1000: LET input=11
00: LET piece=1200: LET flash=13
00: LET charset=1400: LET list=1
500
 369 REM Dibuja las cabeceras de
 las listas de movimientos
 370 PAPER 0: INK 7: PRINT AT 1,
0;"BLANCO": PRINT AT 1,27;"NEGRO
"
 380 PLOT 0,159: DRAW 47,0: DRAW
 0,-1: DRAW -47,0
 390 PLOT 216,159: DRAW 39,0: DR
AW 0,-1: DRAW -39,0
 399 REM Llama a la rutina para
dibujar cada pieza
 400 PAPER 8: FOR K=1 TO 2: FOR
J=1 TO 8: LET I=K: GO SUB piece:
 LET I=9-K: GO SUB piece: NEXT J
: NEXT K
```

```
 500 REM programa principal
 509 REM Matriz de los movimient
os
 510 DIM N$(2,100,5): BORDER 0:
LET S=S1: GO SUB charset: LET N=
1
 519 REM Lee el movimiento blanc
o y hace parpadear el cuadrado e
specificado
 520 LET I$="BLANCAS": GO SUB in
put: LET F=1: GO SUB flash
 529 REM Permite cancelar y camb
iar el movimiento
 530 INPUT "CORRECTO ? ";Y$: IF
Y$<>"S" THEN  LET F=0: GO SUB fl
ash: GO TO 520
 539 REM Mueve la pieza e imprim
e el movimiento
 540 GO SUB move: LET N$(1,N)=M$
: GO SUB list
 549 REM Repite el mismo proceso
 para las negras
 550 LET I$="NEGRAS": GO SUB inp
ut: LET F=1: GO SUB flash
 560 INPUT "CORRECTO ? ";Y$: IF
Y$<>"S" THEN  LET F=0: GO SUB fl
ash: GO TO 550
 570 GO SUB move: LET N$(2,N)=M$
: GO SUB list
 579 REM Siguiente movimiento
 580 LET N=N+1: GO TO 520


1000 REM movimiento
1009 REM Mueve las piezas especi
ficadas
1010 LET B(I2,J2)=B(I1,J1): LET
I=I2: LET J=J2: GO SUB piece
1020 LET B(I1,J1)=0: LET I=I1: L
ET J=J1: GO SUB piece
1030 RETURN

1100 REM entrada
1109 REM Lee las coordenadas del
 cuadrado de origen y el de dest
ino
```

```
1110 INPUT (I$+": JUGADA No."+ST
R$ N+" "); LINE M$;"-"; LINE T$:
 IF M$="STOP" OR M$=" STOP " THE
N  STOP
1120 IF LEN M$<>2 OR LEN T$<>2 T
HEN  GO TO input
1130 LET J1=CODE M$(1)-64: LET I
1=9-(VAL M$(2)): IF I1<1 OR I1>8
 OR J1<1 OR J1>8 THEN  GO TO inp
ut
1140 LET J2=CODE T$(1)-64: LET I
2=9-(VAL T$(2)): IF I2<1 OR I2>8
 OR J2<1 OR J2>8 THEN  GO TO inp
ut
1150 LET M$=M$+"-"+T$: RETURN

1200 REM pieza
1201 REM Datos de entrada I,J
1209 REM Redibuja el cuadrado I,
J con el valor de la matriz del
tablero
1210 LET S=S5: GO SUB charset: L
ET B=B(I,J): INK 0: IF B>10 THEN
  INK 7: LET B=B-10
1220 LET B=2*B: LET A$=CHR$ (63+
B)+CHR$ (64+B): LET B$=CHR$ (95+
B)+CHR$ (96+B)
1230 IF B=0 THEN  LET A$="  ": L
ET B$=A$
1240 PRINT AT I*2+1,J*2+6;A$: PR
INT AT I*2+2,J*2+6;B$
1250 LET S=S1: GO SUB charset: R
ETURN

1300 REM parpadeo
1301 REM Datos de entrada F,I1,J
1,I2,J2
1309 REM Cambia el atributo de p
arpadeo de los cuadrados I1,J1 e
 I2,J2
1310 FLASH F: OVER 1: INK 8: PRI
NT AT I2*2+1,J2*2+6;"  ": PRINT
AT I2*2+2,J2*2+6;"  "
1320 PRINT AT I1*2+1,J1*2+6;"  "
: PRINT AT I1*2+2,J1*2+6;"  "
```

```
1330 FLASH 0: OVER 0: RETURN

1400 REM juego de caracteres
1401 REM Datos de entrada S
1409 REM Cambia la variable del
sistema CHARS al valor de S
1410 LET HI=INT (S/256): LET LO=
S-256*HI
1420 POKE 23606,LO: POKE 23607,H
I: RETURN

1500 REM listado
1509 REM Lista los ultimos 16 mo
vimientos en cada lado de la pan
talla
1510 LET T=N: IF T<16 THEN  LET
T=16
1520 FOR I=T-15 TO T: PRINT AT I
-T+18,0; INK 7;N$(1,I)
1530 PRINT AT I-T+18,27; INK 7;N
$(2,I): NEXT I
1540 RETURN
```

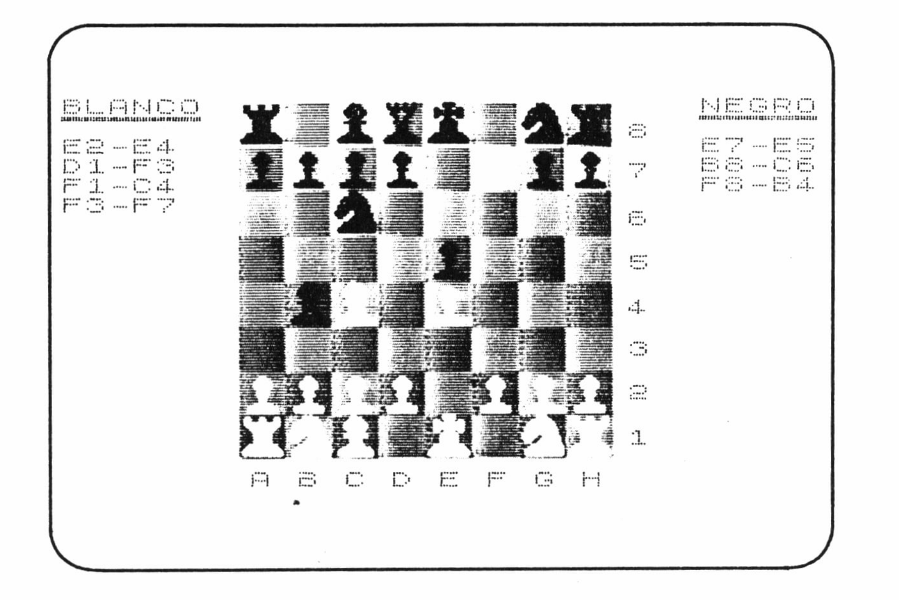


Figura 5.7

Ejercicio 5.8

Adapte el programa del listado 5.7 para que detecte movimientos ilegales y añádale el enroque y la captura de peones "al paso". Si lo suyo ya es *vicio,* añándale las subrutinas necesarias para que el ordenador juegue contra usted.

En el capítulo siguiente consideraremos la combinación de caracteres gráficos y nuestros conocimientos de geometría bidimensional para el trazado de representaciones de datos.

Programas completos

I. Listado 5.1. Datos requeridos: A$. A$ puede ser cualquier carácter salvo los de modo gráfico; pruebe con "X" o "x".

II. Listado 5.2. "Programa principal" y "pixels gigantes" *(main program* y *big pixels).* Datos requeridos: FIL y COL donde $0 \leqslant FIL \leqslant 16$ y $0 \leqslant COL \leqslant 24$. Pruebe con (0, 0) y (1, 1).

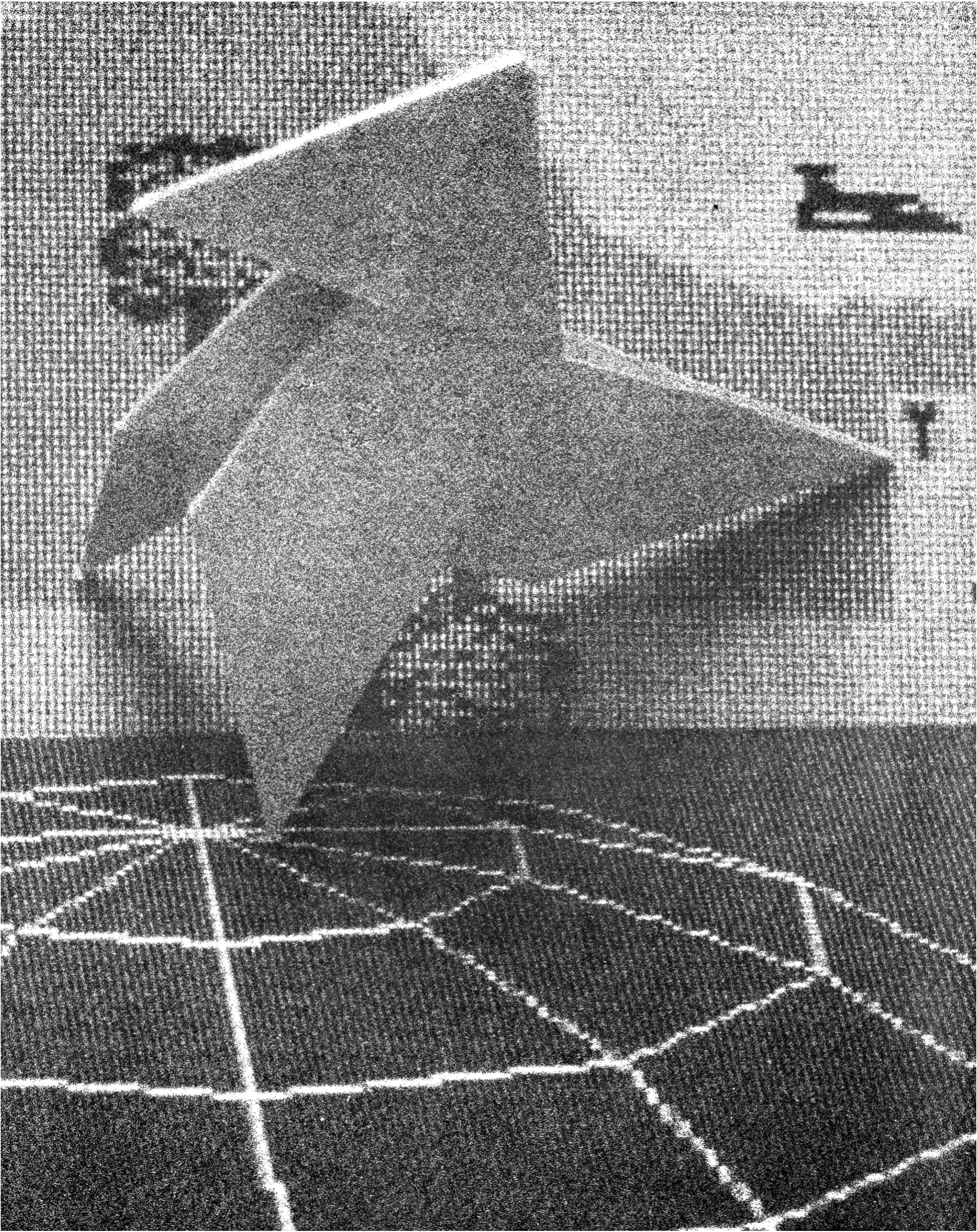
III. Listado 5.3. Datos requeridos: U$ y ocho números BINarios, como máximo de 8 bits, para redefinir el carácter gráfico especificado por U$. Pruebe "A" y 111, 111000, 1010101, 101, 11111001, 1001, 111, 10110; y "B", "C" y "D", cada uno con permutaciones de esos ocho números. A continuación oprima las teclas "A" hasta "U" para ver los caracteres gráficos equivalentes o bien oprima ENTER para volver a empezar el programa.

IV. Listado 5.4: no se precisan datos. El programa III debe emplearse para crear los caracteres gráficos "**A**", "**B**", "**C**" y "**D**".

V. Listado 5.5: el GENERADOR DE CARACTERES. Lea el texto para su descripción y ejemplos de su uso.

VI. Listado 5.6: precisa "master" de la cinta, o bien la generación de sus propios caracteres. Para jugar al "MASTER MIND", teclee el número correspondiente al color de la clavija ("1" a "6") que desee poner. Si teclea "7" elimina la última clavija introducida. Cuando las cuatro estén a su gusto teclee "1" para que el ordenador le conteste.

VII. Listado 5.7: precisa "ajedrez" de la cinta. Teclee las coordenadas de los cuadros de salida y llegada para un movimiento. Las coordenadas de un cuadro vienen dadas por una letra mayúscula ("A" hasta "H") seguida por un número ("1" hasta "8"). Pruebe "E2" seguido de "E4". Para confirmar que es CORRECTO "S", para rechazarlo teclee "N".



6
Diagramas y gráficos de datos

Nuestra época se caracteriza, entre otras cosas, por la cantidad de información producida y consumida: en la actualidad la información es cada vez más abundante y accesible a un mayor número de personas. Un hombre de negocios actual está literalmente abrumado por toneladas de documentos repletos de estadísticas sobre los temas más variopintos, desde capítulos de gastos a estudios de mercado. Los sociólogos nos bombardean constantemente con cifras acerca del desarrollo en la infancia o del aumento registrado en la población de octogenarios en Azerbayjan. Y lo que es peor, los ordenadores nos están inundando a diario con listados de datos mortalmente aburridos sobre cualquier materia, desde astrología a zoología. ¡Evidentemente, habrá que tomar alguna decisión!

Los ordenadores han cooperado en la creación del problema, pero pueden también ayudar a resolverlo. Podemos representar los datos de forma más "digerible" como círculos porcentuales, histogramas, gráficas científicas o diagramas sencillos. Con la llegada de los ordenadores personales se produjo un gran aumento en las ventas de programas que realizaban este tipo de gráficos, hasta tal punto que representa actualmente una de las áreas de mayor crecimiento dentro del campo de los gráficos por ordenador.

En este capítulo comprobaremos que estos diagramas se construyen sin dificultad, disponiendo de unos pocos trucos para mejorar nuestra innata habilidad como delineantes.

Cursores

Ante nosotros tenemos una hoja de papel (PAPER) sobre la que dibujar objetos. Necesitamos un método para controlar con precisión la posición de estos objetos. Generalmente este cometido se realiza con un cursor, controlable externamente por

medio de un "joystick" o cualquier otro periférico de entrada equivalente. Nosotros utilizaremos el teclado como entrada; aunque en principio es más incómodo que el "joystick", presenta la ventaja obvia de no requerir ningún gasto adicional, y los resultados obtenidos a la postre son equivalentes. El subprograma "cursor" del listado 6.1 produce dos líneas cruzadas cuyo punto de corte define el *pixel* o bloque en que se sitúa el cursor. Las líneas están dibujadas con tinta transparente (INK 8) por medio de una orden OVER. El movimiento del cursor se realiza en las ocho direcciones controladas por las teclas situadas alrededor de la "F" (véase figura 6.1). Si tiene un "joystick" u otro periférico similar, modifique el listado 6.1 para introducir la información a través del mismo, en lugar del teclado.

Si pulsa la tecla "F" centrará el cursor en la pantalla. Cuando se mantiene una tecla apretada, el movimiento se hace paulatinamente más rápido. Esta propiedad hace que el programa sea más útil, ya que un movimiento de cursor *pixel* a *pixel* resulta tedioso. Para ayudar a posicionar el cursor, se puede conectar y desconectar una cuadrícula (rejilla) pulsando la letra "L". Al pulsar ENTER para introducir un punto, la cuadrícula desaparece si estaba conectada.

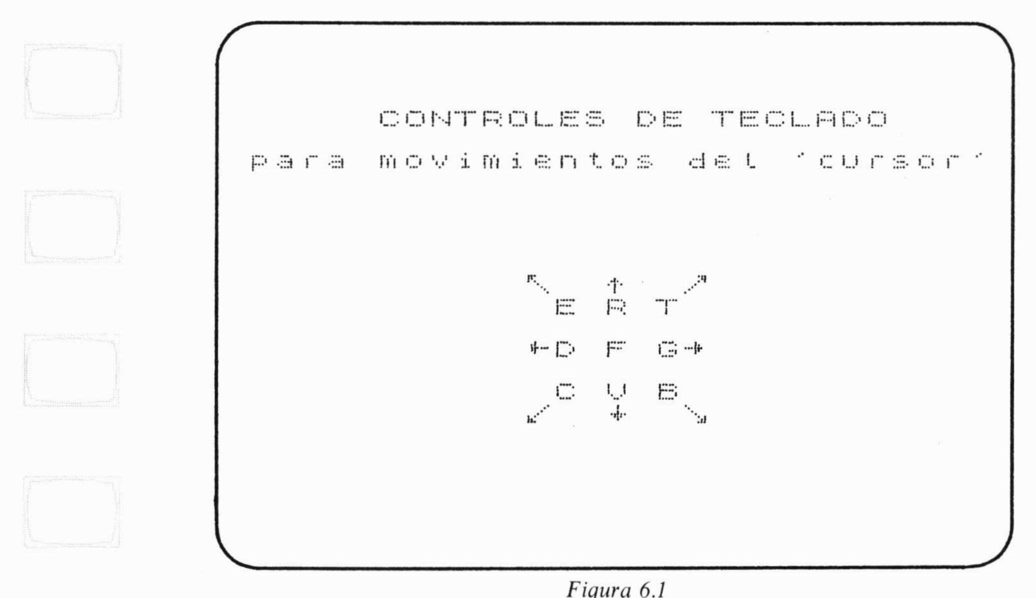


Figura 6.1

Listado 6.1

```
5700 REM subrutina del cursor
5701 REM Datos de entrada PX,PY
5702 REM Datos de salida PX,PY,R
OW,COL
5709 REM La siguiente linea solo
```

```
 es ejecutada en la primera llam
ada a la subrutina
5710 LET PX=128: LET PY=88: LET
cursor=5720: LET grid=5900
5719 REM Comienzo de la rutina d
el cursor
5720 INK 8: LET A=1: LET FLAG=1:
 OVER 1
5729 REM Espera a que se suelten
 las teclas antes de leer el tec
lado
5730 IF INKEY$<>"" THEN  GO TO 5
730
5740 PLOT PX,0: DRAW 0,175: PLOT
 0,PY: DRAW 255,0
5750 LET A$=INKEY$: IF A$="" THE
N  LET A=1: GO TO 5750
5760 IF CODE A$>64 AND CODE A$<9
6 THEN  LET A$=CHR$ (CODE A$+32)
5770 PLOT PX,0: DRAW 0,175: PLOT
 0,PY: DRAW 255,0
5780 IF (A$="e" OR A$="d" OR A$=
"c") AND PX>=A THEN  LET PX=PX-A
5790 IF (A$="c" OR A$="v" OR A$=
"b") AND PY>=A THEN  LET PY=PY-A
5800 IF (A$="e" OR A$="r" OR A$=
"t") AND PY<=175-A THEN  LET PY=
PY+A
5810 IF (A$="t" OR A$="g" OR A$=
"b") AND PX<=255-A THEN  LET PX=
PX+A
5820 IF A$="f" THEN  LET PX=128:
 LET PY=88
5830 IF A$="l" THEN  GO SUB grid
5840 IF A$<>CHR$ 13 THEN  LET A=
A+1: GO TO 5740
5850 LET COL=INT (PX/8): LET ROW
=INT ((175-PY)/8)
5860 IF FLAG=-1 THEN  GO SUB gri
d
5870 INK 0: OVER 0
5880 RETURN

5900 REM rejilla
5909 REM almacena un control par
```

```
a comprobar si la rejilla esta e
n pantalla
5910 FOR J=0 TO 255 STEP 8: PLOT
 J,0: DRAW 0,175: IF J<176 THEN
 PLOT 0,J: DRAW 255,0
5920 NEXT J: LET FLAG=-FLAG: PLO
T 255,0: DRAW 0,175: DRAW -255,0
5930 RETURN
```

Ejercicio 6.1

Prepare un "programa principal" que llame a la subrutina "cursor" y a continuación imprima un cuadrado de color en el bloque especificado. Deberá asimismo imprimir la fila y columna en que ha dibujado (ROW y COL). Cambie la subrutina "cursor" de modo que funcione con las flechas estándar (en las teclas "5", "6", "7" y "8").

Atributos

Podemos aprovechar la idea introducida por el ejercicio 6.1 para fabricar subrutinas que afecten un determinado bloque o conjunto de ellos en sus atributos, sin modificar el contenido del *display file*. El listado 6.2 contiene dos subrutinas que modifican respectivamente los colores de PAPER e INK en un número predeterminado de bloques.

Listado 6.2

```
4000 REM papel
4010 GO SUB cursor: INPUT "COLOR
 ?";P: INPUT "NO. DE BLOQUES (FI
LA*COL) ";R;"*";C
4020 FOR I=ROW TO ROW+R-1: FOR J
=COL TO COL+C-1
4030 PRINT AT I,J; PAPER P; INK
8; OVER 1;" ": NEXT J: NEXT I
4040 RETURN

4100 REM tinta
4110 GO SUB cursor: INPUT "COLOR
 ";P: INPUT "NO. OF BLOQUES (FIL
A*COL) ";R;"*";C
```

```
4120 FOR I=ROW TO ROW+R-1: FOR J
=COL TO COL+C-1
4130 PRINT AT I,J; PAPER 8; INK
P; OVER 1;" ": NEXT J: NEXT I
4140 RETURN
```

Ejercicio ***6.2***

Prepare un "programa principal" que sea capaz de LISTar partes de sí mismo, y a continuación permita destacar partes del listado con diferentes colores de tinta o papel. Escriba instrucciones para modificar los atributos FLASH y BRIGHT de los bloques y colóquelas en las líneas 4050 y 4150 del listado anterior (lo que implica añadir FLASH 8 y BRIGHT 8 a las subrutinas "papel" y "tinta"; intente averiguar por qué). Combine las cuatro subrutinas agrupándolas en una llamada "atributos".

Puntos y líneas

Habiendo ya conseguido un sencillo control interactivo sobre la apariencia de nuestro diagrama, debemos ahora ocuparnos de rellenar los detalles del dibujo. Necesitaremos subprogramas para marcar puntos (PLOT) y dibujar líneas (DRAW). Podríamos utilizar el listado 1.8, pero resultaría tedioso pasar por cada punto de la línea. En su lugar usaremos dos subprogramas: "punto", para marcar *pixels* individuales, y "línea", en el que especificaremos los dos extremos de un segmento para dibujarlo. Los dos subprogramas integran el listado 6.3.

Listado 6.3

```
4200 REM punto
4210 INPUT "ENTER PARA EL CURSOR
"; LINE A$
4220 GO SUB cursor: INPUT "SOBRE
IMPRESION ( 1 O 0 ) ";O: INPUT "
COLOR ";C
4230 PLOT  INK C; OVER O;PX,PY
4240 RETURN

4300 REM linea
4310 INPUT "ENTER PARA EL CURSOR
"; LINE A$
4320 GO SUB cursor: LET SX=PX: L
```

```
ET SY=PY
4330 INPUT "ENTER PARA EL CURSOR
"; LINE A$
4340 GO SUB cursor: INPUT "SOBRE
IMPRESION ( 1 0 0 ) ";O: INPUT "
COLOR ";C
4350 PLOT  PAPER 8; INK C; OVER
O;SX,SY
4360 DRAW  PAPER 8; INK C; OVER
O;PX-SX,PY-SY
4370 RETURN
```

Ejercicio 6.3

Prepare un "programa principal" que le permita utilizar "punto" y "línea" para hacer un dibujo esquemático de su Spectrum. Incorpore "círculo1" o "círculo2" del listado 2.10 en una subrutina "círculo". Los parámetros que necesite, centro y radio, se introducirán usando "cursor". Adapte el programa realizado en el ejercicio 1.3 para emplearlo como subrutina para dibujar un polígono de *n* lados utilizando "cursor" para introducir los puntos. Añada una nueva opción al subprograma "línea" que le permita dibujar líneas curvas, y construya a continuación un diagrama semejante a la figura 1.9 (resultará más sencillo con ocho puntos en el borde en lugar de doce).

Si dispone de un tablero digitalizador (tablilla gráfica) puede copiar dibujos en el tablero introduciéndolos en el ordenador con una versión modificada de "punto" y "línea". Prepare a continuación programas para arreglar el dibujo realizado, es decir, enderezar las líneas rectas y suavizar las curvas.

Grabación y carga de dibujos

Cuando empleamos tiempo y gastamos energías en dibujar y colorear una pantalla, y al final obtenemos el resultado apetecido, puede ser que a continuación deseemos almacenarlo para uso posterior. Evidentemente deberemos poder cargarlo también, condición indispensable para poder alterar con un programa un dibujo realizado por otro. El listado 6.4 contiene las subrutinas "grabación" y "carga" que cumplen ese cometido.

Listado 6.4

```
4500 REM grabacion
4510 INPUT "NOMBRE DEL DIBUJO ?
";N$: IF N$="" THEN  RETURN
```

```
4520 SAVE (N$)SCREEN$
4530 RETURN

4600 REM carga
4610 INPUT "NOMBRE DEL DIBUJO ?
";N$: IF N$="" THEN RETURN
4620 LOAD (N$)SCREEN$
4630 RETURN
```

Ejercicio 6.4

Ejecute el programa del listado 3.1 (o cárguelo con LOAD si lo almacenó previamente); cambie la pantalla a tinta verde sobre papel negro. Además encontrará que hay sentencias semejantes en los dos programas; para ahorrar espacio, combínelos en una sola subrutina.

Etiquetas

Los cuatro listados presentados hasta ahora forman la base de nuestro paquete de construcción de diagramas; sin embargo, no hemos puesto hasta ahora etiquetas en los mismos. Esta tarea la realiza el programa "etiquetado" que se muestra en el listado 6.5. Obsérvese que este listado hace referencia a otros programas no expuestos todavía: esta situación es típica de una construcción de subrutinas en forma modular estructurada, tal como la que hemos adoptado en este libro. En la práctica esto significa que, cuando tropezamos con un problema demasiado grande como para resolverlo al momento, simplemente le asignamos un nombre y lo abordamos más adelante. En el programa "etiquetado" está previsto que las tiras de caracteres pertenecientes al juego número 5 (es decir, aquellas que comiencen por "5:"), se imprimirán en vertical. Por razones que se comentarán más adelante, vamos a reservar un carácter especial (7:) para indicar que la tira de caracteres correspondiente a la etiqueta ha de imprimirse con caracteres más estrechos, utilizando una subrutina especial. Supondremos además que disponemos de una subrutina llamada "set" que cambia el juego de caracteres (esta subrutina se puede encontrar a partir de la instrucción 50 del listado 6.6).

Listado 6.5

```
5400 REM etiquetado
5410 GO SUB cursor: INPUT "JUEGO
:LITERAL ";A$: IF A$="" THEN GO
 TO 5410
5420 INPUT "COLOR ";C$: INK VAL
("0"+C$)
```

```
5430 IF A$( TO 2)="1:" THEN  PRI
NT AT ROW,COL;A$(3 TO ): GO TO 5
510
5439 REM El juego 5 es imprimido
 en vertical
5440 IF A$( TO 2)<>"5:" THEN  GO
 TO 5470
5450 LET S=5: GO SUB set: LET A$
=A$(3 TO )
5460 FOR I=1 TO LEN A$: PRINT AT
 ROW-I+1,COL;A$(I): NEXT I: GO T
O 5510
5469 REM El juego 7 usa la rutin
a de variacion de ancho
5470 IF A$( TO 2)<>"7:" THEN  GO
 TO 5490
5480 LET A$=A$(3 TO ): GO SUB th
in: GO TO 5510
5489 REM Comprueba que se da un
numero valido
5490 LET S=VAL A$(1): IF S>6 OR
S<1 OR A$(2)<>":" THEN  GO TO 55
10
5499 REM Imprime el literal del
juego apropiado
5500 GO SUB set: PRINT AT ROW,CO
L;A$(3 TO )
5510 LET S=1: GO SUB set
5520 INK 0: RETURN
```

Gráficos de datos

Vamos a introducir un "programa principal" de propósito general y una subrutina de selección de opciones, con fin de atar algunos cabos sueltos. Por desgracia, esto hará que aparezcan tantos cabos nuevos como los que acabamos de atar, pero al menos nos permite tener una visión global de las tareas que quedan por resolver. Introducimos identificadores (en minúsculas) para marcar subrutinas que habrá que escribir más adelante, junto con otras cosas, para dibujar tipos especiales de diagramas; por ejemplo, histogramas, círculos porcentuales (familiarmente conocidos como "tartas") y gráficas.

El listado 6.6 lleva incorporada, antes del programa principal, una versión resumida de las partes del generador de caracteres del capítulo 5 que se necesitan para utilizar juegos de caracteres alternativos.

La mayor parte de los diagramas de este libro que no son dibujos generados directamente por los ejemplos han sido realizados con un programa que contiene los listados desde 6.1 a 6.7. Se ha utilizado también para poner etiquetas a muchas figuras producidas por programas; por ejemplo, la figura 4.1.

Listado 6.6

```
   8 REM Para 16K hacer los camb
ios del listado 5.5
   9 REM Inicializacion para usa
r juegos de caracteres alternos
  10 CLEAR 62294
  20 INK 0: PAPER 7: CLS : DIM S
(6)
  30 FOR I=1 TO 6: READ S(I): NE
XT I: DATA 15360,62039,62807,635
75,64343,64848
  40 LET set=50: LET S=1: GO SUB
 set: GO TO 200

  50 REM Cambia al juego S
  60 LET HI=INT (S(S)/256): LET
LO=S(S)-HI*256
  70 POKE 23606,LO: POKE 23607,H
I: RETURN
  80 REM Carga caracteres del ca
sete
  90 INPUT "NOMBRE ? ";N$: IF N$
="" THEN  RETURN
 100 INPUT "NO. DE JUEGO ? ";S:
IF (S<2 OR S>6) AND S<>11 THEN
GO TO 80
 110 INPUT "Arranque el casete y
 de ENTER."; LINE X$
 120 IF S=11 THEN  LOAD N$ CODE
(S(5)+256),768+168: RETURN
 130 IF S=6 THEN  LOAD N$ CODE (
S(S)+512),168: RETURN
 140 LOAD N$ CODE (S(S)+256),768
: RETURN

 200 REM programa principal
 210 LET restart=280: LET query=
500
```

```
 220 LET histo=1000: LET pie=200
0: LET graph=3000
 230 LET paper=4000: LET ink=410
0: LET point=4200: LET line=4300
: LET save=4500: LET load=4600:
LET number=4700
 240 LET charload=80: LET create
=5000: LET thin=5200
 250 LET label=5400: LET cursor=
5700
 260 LET I$="DEFINIR CARACTERES
? ": GO SUB query: IF YES THEN
GO SUB create
 270 CLS : LET I$="CARGAR CARACT
ERES ? ": GO SUB query: IF YES T
HEN  GO SUB charload: GO TO 270
 280 LET I$="CARGAR DIBUJO ? ":
GO SUB query: IF YES THEN  GO SU
B load
 290 LET I$="DIBUJAR DIAGRAMA? "
: GO SUB query: IF NOT YES THEN
 GO TO 360
 300  LET I$="HISTOGRAMA, CIRCUL
O PORC.         O GRAFICA ? ": GO
SUB query
 310 LET diagram=0: IF A$="h" TH
EN  LET diagram=histo
 320 IF A$="c" THEN  LET diagram
=pie
 330 IF A$="g" THEN  LET diagram
=graph
 340 IF diagram=0 THEN  GO TO 29
0
 350 GO SUB diagram
 360 LET I$="ETIQUETAR DIBUJO ?
": GO SUB query: IF YES THEN  GO
 SUB label: GO TO 360
 370 LET I$="CAMBIAR COLOR PAPEL
 ? ": GO SUB query: IF YES THEN
 GO SUB paper: GO TO 370
 380 LET I$="CAMBIAR COLOR TINTA
 ? ": GO SUB query: IF YES THEN
 GO SUB ink: GO TO 380
 390 LET I$="DIBUJAR UN PUNTO? "
: GO SUB query: IF YES THEN  GO
```

```
SUB point: GO TO 390
 400 LET I$="DIBUJAR LINEA? ": G
O SUB query: IF YES THEN  GO SUB
 line: GO TO 400
 410 LET I$="TERMINAR DIBUJO ? "
: GO SUB query: IF NOT YES THEN
 GO TO restart
 420 LET I$="SALVAR DIBUJO ? ":
GO SUB query: IF YES THEN  GO SU
B save
 430 STOP

 500 REM opciones
 510 LET YES=0
 520 INPUT (I$); LINE A$: IF A$=
"" THEN  RETURN
530 LET A$=A$(1): IF CODE A$<96
THEN  LET A$=CHR$ (CODE A$+32)
540 LET YES=(A$="s")
550 RETURN
```

Caracteres especiales

Para completar nuestros preparativos para la subrutina "etiquetado" (*label,* listado 6.5), necesitamos crear un juego de caracteres girados, que colocaremos en el juego número 5 para escribir etiquetas verticales. Esta tarea está asignada al subprograma "crear" del listado 6.7, que utiliza algunas técnicas del generador de caracteres para copiar una versión girada del juego 1 en el juego 5.

Listado 6.7

```
5000 REM crear/caracteres
5009 REM Crea caracteres para la
 rutina del histograma
5010 DIM P(8): DIM D(3): LET D(1
)=15: LET D(2)=255: LET D(3)=240
5020 FOR I=0 TO 6: FOR J=0 TO 7
5030 LET P(1)=USR "A"+I*8+J: LET
 P(2)=USR "H"+I*8+J
5040 LET P(3)=USR "O"+I*8+J
5050 FOR K=1 TO 3: POKE P(K),0:
```

```
NEXT K
5060 IF J>I THEN  FOR K=1 TO 3:
POKE P(K),D(K): NEXT K
5070 NEXT J: NEXT I
5078 REM Copia el juego 1 en el
5 con los caracteres rotados  .
5079 REM Para 16K Q=31831
5080 DIM B$(8,8): LET T=256: FOR
 I=1 TO 8: LET T=T/2: LET P(I)=T
: NEXT I: LET P=15616: LET Q=645
99
5090 FOR I=32 TO 127: FOR J=1 TO
 8: LET B$(J)="00000000": NEXT J
5100 FOR J=1 TO 8: LET T=PEEK P:
 FOR K=1 TO 8
5110 IF T>=P(K) THEN  LET T=T-P(
K): LET B$(9-K,J)="1"
5120 NEXT K: LET P=P+1: NEXT J
5130 FOR J=1 TO 8: POKE Q,VAL ("
BIN "+B$(J)): LET Q=Q+1: NEXT J
5140 PRINT AT 10,10;"ROTANDO ";C
HR$ I
5150 NEXT I: CLS
5160 RETURN

5200 REM variacion de ancho
5201 REM Datos de entrada ROW,CO
L,A$
5209 REM Imprime cualquier otro
caracter en la mitad izquierda d
e los bloques
5210 LET S=4: GO SUB set: OVER 1
: LET TC=COL
5220 FOR I=1 TO LEN A$ STEP 2: P
RINT AT ROW,TC;A$(I): LET TC=TC+
1: NEXT I
5229 REM Imprime los otros carac
teres en la mitad derecha
5230 LET S=3: GO SUB set: LET TC
=COL
5240 FOR I=2 TO LEN A$ STEP 2: P
RINT AT ROW,TC;A$(I): LET TC=TC+
1: NEXT I
5250 LET S=1: GO SUB set: OVER 0
5260 RETURN
```

Antes de generar los caracteres, el programa redefine el conjunto de caracteres gráficos definidos por el usuario haciéndolo tres conjuntos de siete caracteres que se pueden observar en la figura 6.2.



Figura 6.2

De este modo se puede conseguir rellenar de tinta la mitad derecha o izquierda del bloque, o el bloque completo, comenzando desde abajo. Con estos caracteres se pueden conseguir líneas horizontales de diferentes grosores; sin embargo, su uso principal es el trazado de histogramas.

De aquí al final del capítulo discutiremos algunos tipos comunes de diagramas. La variación existente es enorme, por lo que no resulta práctico abordar todas las posibilidades. Por consiguiente, estudiaremos detalladamente unos cuantos ejemplos representativos de los tres tipos principales de gráficos: histogramas, círculos porcentuales y gráficas. A partir de ahí se pueden diseñar variaciones que permitan presentar los datos en la forma que se desee. Obsérvese de nuevo que los ordenadores de 16K han de introducir cambios comentados en el Apéndice A.

Histogramas

Los histogramas (también llamados diagramas de barras) pueden construirse con nuestros programas de cualquier altura y con *pixels* de cualquier color, a condición de que las barras tengan al menos medio bloque de anchura. Como esto será lo usual, expondremos un método para calcular el espaciado y anchura de las barras, una vez conocido el número de éstas que se desea dibujar. La primera parte del

listado “histograma“ (listado 6.8) dibuja los ejes, etiqueta el vertical y pregunta cuántas barras se desea dibujar. A continuación se calcula la anchura de las barras y espacios intermedios (GAP) como múltiplos de medios bloques, a través de un método que asegura que 1/2 ⩽ GAP ⩽ X. Al ir introduciendo los datos, el programa utiliza los caracteres de bloque y medio bloque, junto con el juego de caracteres creado anteriormente, y con ellos construye tiras *(strings)* que representen la altura total de cada barra. A la tira conseguida se le antepone “5:”, y se usa la subrutina “barra” para imprimirla verticalmente. Obsérvese que los ejes se sitúan en los bloques adyacentes a los de comienzo de las barras, con el fin de garantizar la independencia de color de unas y otros.

Listado 6.8

```
1000 REM histograma/tipo 1
1009 REM Iguala el puntero a la
subrutina que imprime barras
1010 LET bar=1320
1019 REM Halla la escala y dibuj
a los ejes
1020 INPUT "RANGO VERTICAL ";YB;
" A ";YT
1030 IF YB>=YT THEN  GO TO 1020
1040 LET YSCALE=128/(YT-YB)
1050 PLOT 47,152: DRAW 0,-129: D
RAW 201,0
1059 REM Etiqueta el eje vertica
l
1060 LET YDIF=(YT-YB)/4: LET TIC
K=YB
1070 FOR I=1 TO 5: LET TK=INT (T
ICK+0.5)
1080 LET Y=32*I-8: PLOT 47,Y: DR
AW -3,0: LET ROW=INT ((176-Y)/8)
-1
1090 LET A$=STR$ (TK): IF LEN A$
>3 THEN  LET A$=A$( TO 3): IF TK
>999 OR TK<-99 THEN  LET A$="***
"
1100 LET COL=1: IF TK>=0 THEN  L
ET COL=COL+1
1110 IF ABS TK<10 THEN  LET COL=
COL+1
1120 IF ABS TK<100 THEN  LET COL
=COL+1
```

```
1130 PRINT AT ROW,COL;A$: LET TI
CK=TICK+YDIF
1140 NEXT I
1149 REM Lee el numero de barras
 requerido
1150 INPUT "No. DE BARRAS ";NB
1160 LET X=INT (25/NB+0.5)/2: LE
T GAP=INT (50/NB-2*X)/2
1170 LET GP=(25-NB*X-(NB-1)*GAP)
*2: LET GP=INT ((GP+1)/2)/2
1179 REM Posiciona el puntero de
 columna en la barra 1 y repite
el bucle para cada barra
1180 LET COL=6+GP: FOR I=1 TO NB
1190 LET Y$="DATOS PARA BARRA "+
STR$ I+" ": INPUT (I$);D;" COLOR
 ";C: INK C: LET W=X
1199 REM Los valores por debajo
del eje horizontal tienen altura
 minima
1200 IF D<=YB THEN  LET D=0: LET
 H=1: GO TO 1220
1209 REM Calcula el numero de bl
oques de altura para la barra
1210 LET D=D-YB: LET H=INT (D*YS
CALE+0.5)
1220 LET IH=INT (H/8): LET L$=""
: LET M$="": LET R$=""
1229 REM Construye literales par
a las barras de la izq., medio,
y derecha
1230 FOR J=1 TO IH: LET L$=L$+CH
R$ 133: LET M$=M$+CHR$ 143: LET
R$=R$+CHR$ 138
1239 REM Halla el no. de element
os de pantalla para el caracter
especial superior
1240 NEXT J: LET RH=H-IH*8: IF R
H=0 THEN  GO TO 1260
1249 REM Pone caracteres especia
les en la parte superior de las
barras
1250 LET L$=L$+CHR$ (151-RH): LE
T M$=M$+CHR$ (158-RH): LET R$=R$
+CHR$ (165-RH)
```

```
1260 IF COL=INT (COL) THEN  GO T
O 1280
1270 LET A$=L$: GO SUB bar: LET
W=W-0.5: LET COL=COL+0.5
1279 REM Imprime el numero de bl
oques completos necesarios   .
1280 IF W>=1 THEN  LET A$=M$: GO
 SUB bar: LET W=W-1: LET COL=COL
+1: GO TO 1280
1290 IF W>0.1 THEN  LET A$=R$: G
O SUB bar: LET COL=COL+0.5
1299 REM Mueve el puntero de col
umna a la siguiente barra y rein
icia el bucle
1300 LET COL=COL+GAP: NEXT I
1310 INK 0: RETURN

1320 REM barra
1321 REM Datos de entrada COL,A$
1329 REM Imprime A$ verticalment
e desde la columna 19 en adelant
e
1330 FOR K=1 TO LEN A$: PRINT AT
 19-K,INT COL;A$(K): NEXT K
1340 RETURN
```

La figura 6.3, un diagrama que representa la lluvia mensual promedio en Cantabreda, ha sido realizado con el subprograma “histograma/tipo 1” del listado 6.8 seguido del programa de etiquetado para los rótulos.

Ejercicio 6.5

Introduzca variaciones en el programa “histograma” por medio de subrutinas que puedan integrarse a voluntad con una orden MERGE. Escriba una subrutina que calcule la posición de parejas de barras, donde las barras de la misma pareja estén separadas por medio bloque, en tanto que las parejas de barras adyacentes disten entre sí al menos un bloque. Utilícela para dibujar diagramas semejantes al de la figura 6.4.

Un ejemplo de sustitución de “histograma/tipo 1” lo tenemos en el listado 6.9, llamado “histograma/tipo 2”. En este caso el cálculo de la anchura de las barras se modifica para conseguir valores enteros de bloques para X y GAP. Se necesitan dos datos para cada barra, que especifican el rango de altura MAXima y MINima de

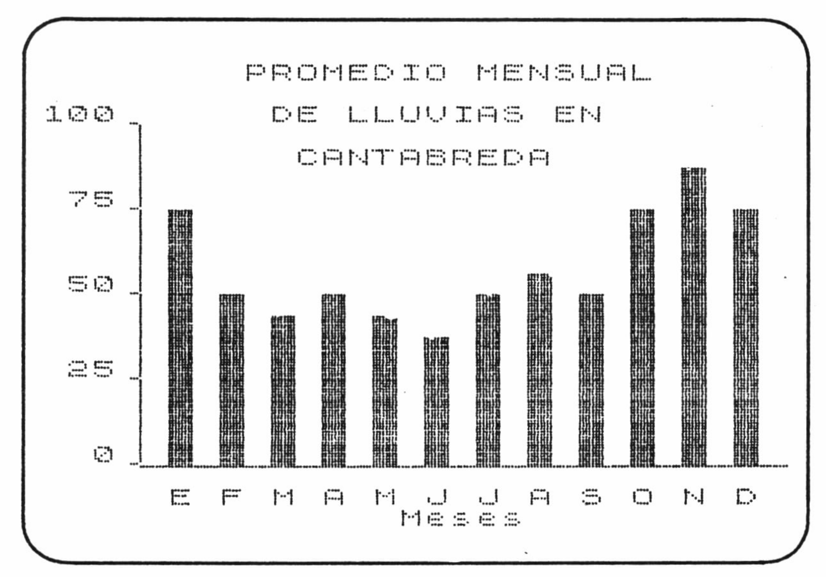


Figura 6.3

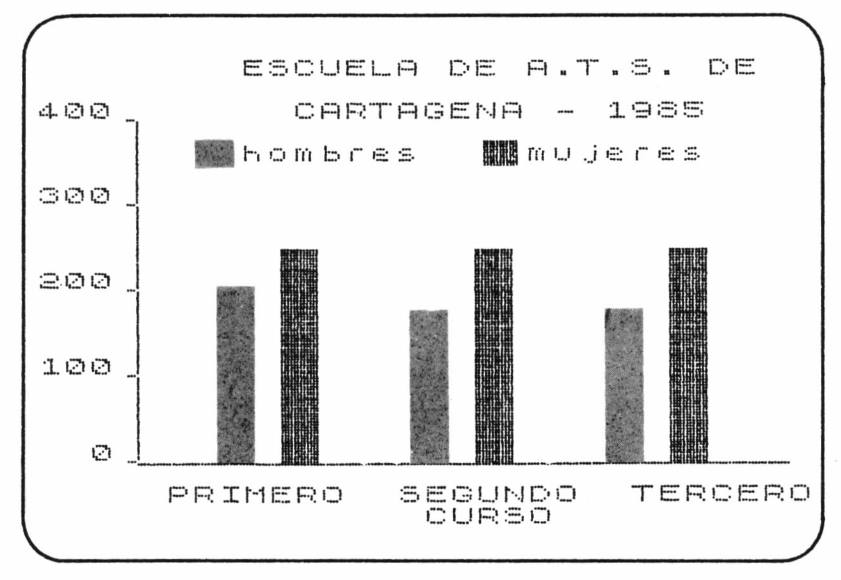


Figura 6.4

la misma. Se pueden conseguir resultados como el de la figura 6.5, por medio de órdenes OVER. Si desea comprender en profundidad el funcionamiento del programa (y de otros programas de este libro), es aconsejable distribuir hábilmente órdenes PRINT por el listado, de forma que se pueda seguir la lógica de los algoritmos a medida que se van ejecutando.

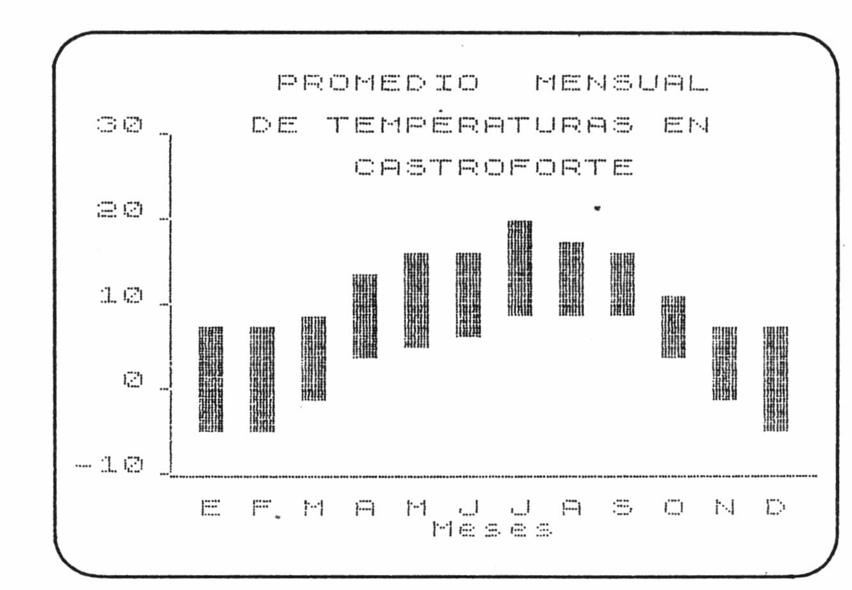


Figura 6.5

Listado 6.9

```
1000 REM histograma/tipo 2
1009 REM Fija el puntero a la ru
tina que imprime barras
1010 LET bar=1320: DIM O$(2,3):
LET O$(1)="MAX": LET O$(2)="MIN"
1019 REM Halla la escala y dibuj
a los ejes
1020 INPUT "RANGO VERTICAL ";YB;
" A ";YT
1030 IF YB>=YT THEN  GO TO 1020
1040 LET YSCALE=128/(YT-YB)
1050 PLOT 47,152: DRAW 0,-129: D
RAW 201,0
1059 REM Etiqueta el eje vertica
l
1060 LET YDIF=(YT-YB)/4: LET TIC
K=YB
1070 FOR I=1 TO 5: LET TK=INT (T
ICK+0.5)
1080 LET Y=32*I-8: PLOT 47,Y: DR
AW -3,0: LET ROW=INT ((176-Y)/8)
-1
```

```
1090 LET A$=STR$ (TK): IF LEN A$
>3 THEN  LET A$=A$( TO 3): IF TK
>999 OR TK<-99 THEN  LET A$="***
"
1100 LET COL=1: IF TK>=0 THEN  L
ET COL=COL+1
1110 IF ABS TK<10 THEN  LET COL=
COL+1
1120 IF ABS TK<100 THEN  LET COL
=COL+1
1130 PRINT AT ROW,COL;A$: LET TI
CK=TICK+YDIF
1140 NEXT I
1149 REM Lee el no. de barras re
querido y calcula un ancho enter
o para ellas
1150 INPUT "No. DE BARRAS ";NB:
OVER 1
1160 LET GAP=INT (13/NB): LET X=
INT ((26-GAP*NB)/NB)
1170 LET GP=(25-NB*X-(NB-1)*GAP)
: LET GP=INT ((GP+1)/2)
1179 REM Posiciona el puntero de
 columna en la primera barra y r
epite el bucle
1180 LET COL=6+GP: FOR I=1 TO NB
: INPUT ("COLOR PARA BARRA "+STR
$ (I)+": ");C: INK C
1189 REM Bucle de entrada de val
ores maximos y minimos para barr
as
1190 FOR O=1 TO 2: LET I$=O$(O)+
" PARA BARRA "+STR$ I+" ": INPUT
 (I$);D: LET W=X
1200 IF D<=YB THEN  LET D=0: LET
 H=1: GO TO 1220
1209 REM Calcula el numero de bl
oques completos para las barras
1210 LET D=D-YB: LET H=INT (D*YS
CALE+0.5)
1220 LET IH=INT (H/8): LET M$=""
1229 REM Construye una cadena pa
ra las barras
1230 FOR J=1 TO IH: LET M$=M$+CH
R$ 143
```

```
1239 REM Imprime un caracter esp
ecial para la altura remanente
1240 NEXT J: LET RH=H-IH*8: IF R
H=0 THEN  GO TO 1260
1250 LET M$=M$+CHR$ (158-RH)
1260 IF O=1 THEN  LET OCOL=COL
1270  IF O=2 THEN  LET COL=OCOL
1279 REM Imprime el numero de bl
oques de ancho completo requerid
o
1280 IF W>=1 THEN  LET A$=M$: GO
 SUB bar: LET W=W-1: LET COL=COL
+1: GO TO 1280
1290 LET COL=COL+GAP: NEXT O: NE
XT I
1300 INK 0: OVER 0: RETURN

1320 REM barra
1321 REM Datos de entrada COL1A$
1329 REM Imprime A$ verticalment
e de la columna A$ en adelante
1330 FOR K=1 TO LEN A$: PRINT AT
 19-K,INT COL;A$(K): NEXT K
1340 RETURN
```

Una buena cosa del BASIC es que permite introducir sentencias STOP en un programa, interrogar al ordenador acerca de los valores de las variables y CONTinuar la ejecución sin afectar el programa. Este sistema es muy útil para corregir programas, así como para comprender listados ajenos.

Ejercicio 6.6

Intente utilizar distintos colores de papel e imprima las barras con OVER a fin de conseguir barras de dos colores. Tenga cuidado cuando utilice datos cuya diferencia de altura sea inferior a ocho *pixels.*

Cuando se usan dos colores diferentes para cada barra (ejercicio 6.6) pueden surgir problemas: concretamente, si el valor MACimo y MINimo caen en el mismo bloque, dicho bloque tendrá asignados tres colores, los dos de la barra y el fondo. Se puede obviar el problema comparando la longitud de las tiras *(strings)* que contienen los caracteres para MAX y MIN: si son de la misma longitud, se trunca MIN eliminando el último carácter. Este procedimiento evita el problema de los colores pero resulta ligeramente inexacto. Otra forma de evitarlo es asignar a la parte

inferior de la barra el mismo color que al fondo, lo que garantiza que la barra se pueda dibujar con exactitud con dos colores como máximo por bloque.

Hay una enorme cantidad de variaciones sobre el tema; por ejemplo, dibujar barras por encima y por debajo de una línea central para mostrar fluctuaciones relativas (por ejemplo, en cambio de divisas). Con las ideas apuntadas deberá ser capaz de producir el tipo de histograma que se ajuste más a sus propias especificaciones.

Círculos porcentuales

Los círculos porcentuales o diagramas de "tarta" son los preferidos por los economistas y biólogos, unos explicándonos quién o qué se lleva la mayor tajada del pastel de gastos, los otros mostrando qué bacteria se lo está comiendo. A un programa de este tipo se le exigirá normalmente que sea capaz de dibujar círculos de radio variable, que se pueda sacar un trozo de tarta para mostrarlo por separado, y que permita colorear o sombrear las distintas porciones. La subrutina "círculo porcentual" *(pie)* dada en el listado 6.10 realiza las dos primeras tareas utilizando "cursor" para centrar el diagrama, e introduciendo el radio en *pixels* con un INPUT.

Listado 6.10

```
2000 REM circulo porcentual
2009 REM Fija los punteros de la
s rutinas
2010 LET hatch=2300: LET in=2800
2019 REM Se leen todos los segme
ntos para calcular la escala ang
ular
2020 INPUT "No. DE SEGMENTOS ";N
B: INPUT "COLOR ";C$: LET C=VAL
("0"+C$): LET TOT=0
2030 DIM D(NB): FOR I=1 TO NB: I
NPUT ("DATO "+STR$ I+": ");D(I):
 LET TOT=TOT+D(I): NEXT I
2039 REM Usa el cursor para espe
cificar el centro del circulo
2040 INPUT "ENTER PARA CENTRAR e
L CIRCULO"; LINE Y$
2050 GO SUB cursor: LET XC=PX: L
ET YC=PY
2060 INPUT "RADIO (EN PIXELS) ";
RAD
```

```
2070 LET ASCALE=2*PI/TOT: LET A1
=PI/2
2080 FOR I=1 TO NB
2090 INPUT "ENTER PARA CENTRAR P
ORCION"; LINE Y$
2100 LET PX=XC: LET PY=YC: GO SU
B cursor: LET ANG=ASCALE*D(I): L
ET A2=A1-ANG
2109 REM Si se mueve la porcion
calcula el desplazamiento en su
bisectriz
2110 IF PX=XC AND PY=YC THEN  GO
 TO 2140
2120 LET A3=A1-ANG/2: LET DIST=S
QR ((PXmXC)*(PX-XC)+(PY-YC)*(PY-
YC))
2130 LET PX=INT (XC+DIST*COS A3+
0.5): LET PY=INT (YC+DIST*SIN A3
+0.5)
2139 REM Comprueba si se requier
e sombreado y de que tipo
2140 INPUT "SOMBREADO (x,y,xy,n)
 ";H$: IF CODE H$<96 THEN  LET H
$=CHR$ (CODE H$+32)
2149 REM Pregunta la distancia e
ntre lineas y el desplazamiento
2150 IF H$<>"n" THEN  INPUT "INC
. ";JUMP,"DESP. ";REM
2159 REM Dibuja el segmento del
circulo
2160 INK C: PLOT PX,PY: LET X1=I
NT (RAD*COS A1+0.5): LET Y1=INT
(RAD*SIN A1+0.5): DRAW X1,Y1
2170 LET A2S=A2: LET ASTO=ANG: L
ET XA=PX+X1: LET YA=PY+Y1
2180 LET XB=INT (RAD*COS A2+0.5)
: LET YB=INT (RAD*SIN A2+0.5)
2190 FOR T=A1 TO A2 STEP -3/RAD
2200 LET X2=INT (RAD*COS T+0.5):
 LET Y2=INT (RAD*SIN T+0.5): DRA
W X2-X1,Y2-Y1: LET X1=X2: LET Y1
=Y2
2210 NEXT T: DRAW XB-X2,YB-Y2: L
ET X2=XB: LET Y2=YB
2220 DRAW -XB,-YB: IF H$="n" THE
```

```
N  GO TO 2250
2228 REM Llama a la rutina de so
mbreado si se necesita
2229 REM Si el angulo es >PI hac
e el sombreado en dos partes
2230 IF ASTO>PI THEN  LET A2=A1-
PI: LET XB=2*PX-XA: LET YB=2*PY-
YA: GO SUB hatch: LET XA=XB: LET
 YA=YB: LET A1=A2: LET A2=A2S
2240 LET XB=PX+X2: LET YB=PY+Y2:
 GO SUB hatch
2249 REM Reinicia el bucle para
el siguiente segmento
2250 LET A1=A2: NEXT I: RETURN
```

A continuación se introducen los datos individuales por INPUT y se calcula el total, TOT. Este último valor se utiliza para calcular una escala en radianes para dividir la tarta. Cada trozo se centra con el cursor, tomando los desplazamientos de éste como distancias en la bisectriz de la porción, no como posiciones absolutas. Con cada nueva porción el "cursor" reaparece en el centro original de la tarta. La figura 6.6 se ha realizado utilizando esta subrutina.

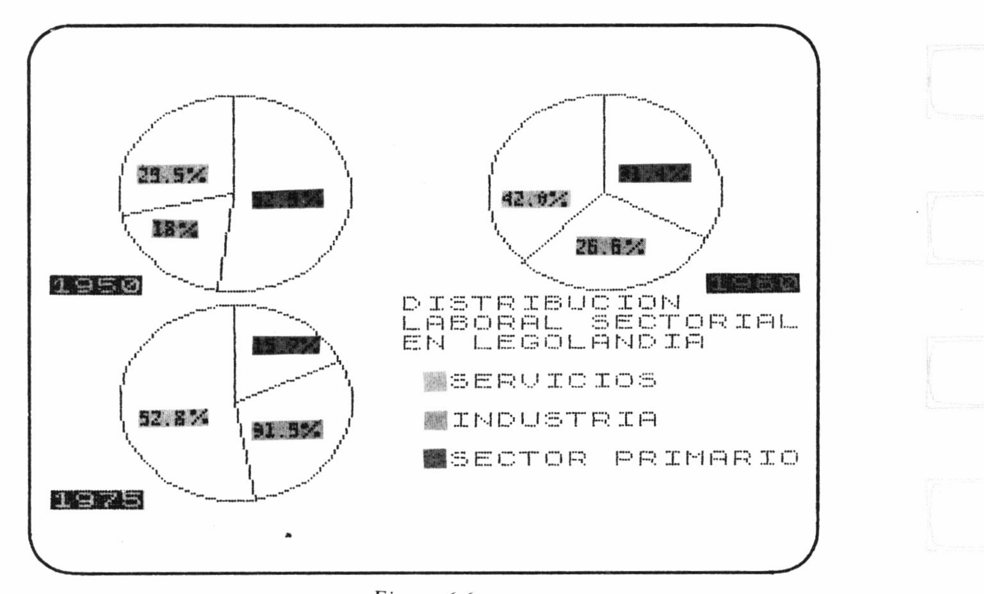


Figura 6.6

Sombreado

Un sombreado por trozos de un diagrama tipo tarta implica intersecciones de líneas con los límites de cada porción. Para simplificar los cálculos sombrearemos con líneas verticales y horizontales únicamente, pudiendo por tanto utilizar en una porción determinada líneas verticales únicamente, líneas horizontales o bien una mezcla de ambas. Además sombrearemos siempre porciones menores de π radianes (180 grados). Los trozos mayores se consideran como dos porciones independientes, el primero de π radianes y el otro del resto, y se sombrearán por separado.

La subrutina "círculo porcentual" pregunta si se desea en el trozo un sombreado horizontal (respuesta "x"), vertical (respuesta "y"), ambos (respuesta "xy") o ninguno (respuesta "n").

Cada uno de los trozos de tarta a sombrear está circundado por dos segmentos rectilíneos y un arco circular. Debemos encontrar la parte de la línea de sombreado que cae dentro de esta superficie (suponiendo que exista tal parte). Como la máxima apertura angular a sombrear son π radianes, las únicas posibilidades que aparecen son:

1) la línea de sombreado no corta la tarta (y por tanto no es tal);
2) la línea corta el arco en dos puntos;
3) la línea corta el arco y uno de los segmentos;
4) la línea corta los dos segmentos.

Hay casos particulares, como que la línea corte precisamente en la intersección entre el arco y un segmento; de cualquier forma, estos casos pueden englobarse a nuestros efectos en uno de los cuatro ya apuntados.

Explicamos el algoritmo de sombreado con líneas horizontales: el razonamiento es idéntico para verticales. En primer lugar localizamos los valores MAXimo y MINimo de y que acotan la porción de tarta. A continuación consideramos todas las líneas horizontales de sombreado como rectas de la forma $Y = k*\text{JUMP} + \text{REM}$ entre los límites $0 \leqslant \text{REM} \leqslant \text{JUMP} - 1$. En cada recta calcularemos los dos puntos de intersección con los segmentos prolongados, y comprobaremos que su valor MU está comprendido entre 0 y 1, es decir, que la intersección está comprendida entre el centro del círculo y el arco. Seguidamente buscaremos los dos puntos de intersección de la línea de sombreado con la circunferencia a la que pertenece el arco. A partir de estas intersecciones seleccionaremos los dos puntos de corte del trozo de la tarta con la línea de sombreado, y los uniremos con un segmento.

El proceso completo está programado en la subrutina del listado 6.11. Un resultado típico es el que se muestra en la figura 6.7. Obsérvese que para rellenar por completo la porción basta con hacer JUMP igual a 1.

Listado 6.11

```
2300 REM sombreado
2309 REM Si se requiere sombread
o cruzado ejecuta la rutina dos
veces
```

```
2310 IF H$="xy" THEN  LET H$="x"
: GO SUB 2320: LET H$="y": GO SU
B 2320: LET H$="b": RETURN
2319 REM Fija las variables de s
ombreado para la direccion de la
s lineas
2320 IF H$="y" THEN  LET PZ=PX:
LET PT=PY: LET ZA=XA: LET TA=YA:
 LET ZB=XB: LET TB=YB
2330 IF H$="x" THEN  LET PZ=PY:
LET PT=PX: LET ZA=YA: LET TA=XA:
 LET ZB=YB: LET TB=XB
2340 DIM Z(3)
2349 REM Halla las coordenadas m
ax. y min. de las lineas dentro
del segmento
2350 LET T=PI/2: LET MAX=0: LET
MIN=0
2360 LET VAL=SIN A1: IF H$="x" T
HEN  LET VAL=COS A1
2370 IF MAX<VAL THEN  LET MAX=VA
L
2380 IF MIN>VAL THEN  LET MIN=VA
L
2390 IF T>A1 THEN  LET T=T-PI/2:
 GO TO 2390
2400 IF T<A2 THEN  GO TO 2450
2410 LET VAL=SIN T: IF H$="x" TH
EN  LET VAL=COS T
2420 IF MAX<VAL THEN  LET MAX=VA
L
2430 IF MIN>VAL THEN  LET MIN=VA
L
2440 LET T=T-PI/2: GO TO 2400
2450 LET VAL=SIN A2: IF H$="x" T
HEN  LET VAL=COS A2
2460 IF MAX<VAL THEN  LET MAX=VA
L
2470 IF MIN>VAL THEN  LET MIN=VA
L
2480 LET NEWMIN=INT (INT (RAD*MI
N+1)/JUMP)*JUMP+REM
2489 REM Encuentra la intersecci
on de las lineas con el radio y
el arco
```

```
2490 FOR E=NEWMIN TO MAX*RAD STE
P JUMP
2499 REM Almacena las coordenada
s de las intersecciones en la ma
triz Z
2500 LET IC=0
2510 LET DENOM=TA-PT: IF DENOM=0
 THEN  GO TO 2540
2520 LET MU=E/DENOM: IF MU<0 OR
MU>1 THEN  GO TO 2540
2530 LET IC=IC+1: LET Z(IC)=PZ+M
U*(ZA-PZ)
2540 LET DENOM=TB-PT: IF DENOM=0
 THEN  GO TO 2580
2550 LET MU=E/DENOM: IF MU<0 OR
MU>1 THEN  GO TO 2580
2560 LET IC=IC+1: LET Z(IC)=PZ+M
U*(ZB-PZ)
2569 REM Anula los puntos de int
erseccion duplicados
2570 IF IC=2 AND Z(1)=Z(2) THEN
 LET IC=1
2580 IF IC<>2 THEN  GO TO 2610
2589 REM Dibuja las lineas de so
mbreado
2590 IF H$="y" THEN  PLOT Z(1),E
+PT: DRAW Z(2)-Z(1),0: GO TO 271
0
2600 IF H$="x" THEN  PLOT E+PT,Z
(1): DRAW 0,Z(2)-Z(1): GO TO 271
0
2610 LET DISC=RAD*RAD-E*E: IF DI
SC<0 THEN  GO TO 2710
2620 LET DISC=INT (SQR DISC+0.5)
2630 LET ZZ=PZ+DISC: LET AZ=DISC
: GO SUB in: IF OUT THEN  GO TO
2650
2640 LET IC=IC+1: LET Z(IC)=ZZ
2650 LET ZZ=PZ-DISC: LET AZ=-DIS
C: GO SUB in: IF OUT THEN  GO TO
 2670
2660 LET IC=IC+1: LET Z(IC)=ZZ
2670 IF IC<2 THEN  GO TO 2710
2680 IF IC=2 THEN  GO TO 2590
2690 IF Z(1)=Z(2) THEN  LET Z(2)
```

```
=Z(3)
2700 GO TO 2590
2710 NEXT E
2720 RETURN
2800 REM dentro

2808 REM Calcula el angulo del c
entro al punto de interseccion
2809 REM Si esta entre los ang.
min. y max. del segmento el punt
o esta en el arco
2810 LET BZ=AZ: LET EZ=E
2820 IF H$="x" THEN  LET BZ=E: L
ET EZ=AZ
2830 IF BZ=0 THEN  LET PHI=-PI/2
: IF EZ>0 THEN  LET PHI=-PHI
2840 IF BZ=0 THEN  GO TO 2860
2850 LET PHI=ATN (EZ/BZ): IF BZ<
0 THEN  LET PHI=PHI-PI
2859 REM Fija un control para in
dicar si el punto es valido
2860 LET OUT=(PHI>=A1) OR (PHI<=
A2)
2870 RETURN
```

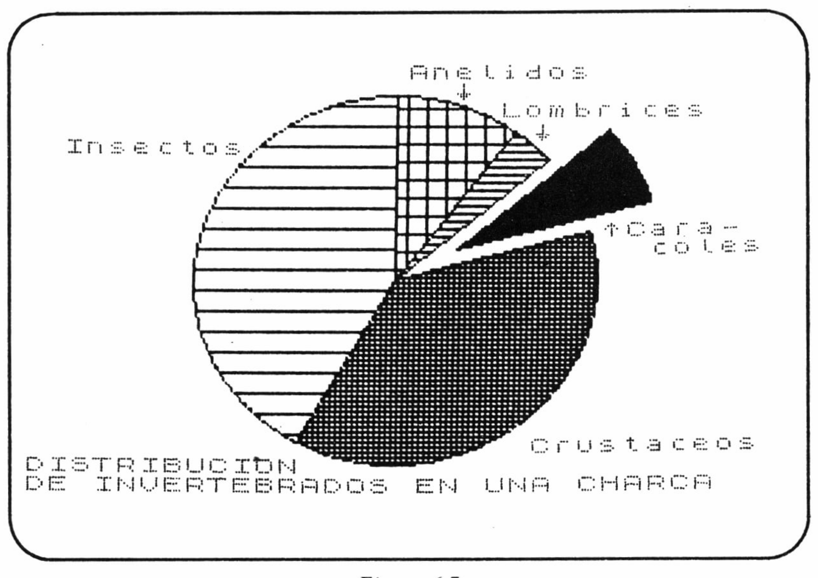


Figura 6.7

Como último ejemplo de representación de gráficos consideraremos el trazado de gráficas de funciones o de puntos discretos manejados frecuentemente en el mundo científico. En tales diagramas deberán incluirse ambos ejes coordenados, cuyas escalas pueden cubrir una variedad extraordinaria de rangos y cuyo origen deberá ser flotante. El método que usaremos para decidir dónde colocaremos un determinado eje es bastante estándar: si en el rango cubierto por el eje se encuentra el cero de la gráfica, entonces el eje pasará por ese punto; si por el contrario el cero quedara fuera de la gráfica, colocaríamos el eje sobre el borde del área del gráfico que está más cercano al cero. Sobre cada uno de los ejes se colocan cinco pequeños guiones perpendiculares al eje (TICK), y el valor correspondiente a cada TICK se colocará próximo al mismo. Con el fin de obtener gráficas científicas precisas, deberíamos incluir el mayor número posible de caracteres posibles. Tal como están las cosas, sólo podemos colocar 32 caracteres en dirección horizontal y 22 en vertical. Aquí es donde actúa "variación de ancho". Anteriormente, habíamos empleado el GENERADOR DE CARACTERES para crear dos juegos de caracteres reducidos con la mitad de anchura de los normales: un juego que los tiene colocados en la mitad izquierda del bloque del carácter, y el otro juego a la derecha. Cuando queremos imprimir una tira de caracteres en la forma reducida, la subrutina "variación de ancho" imprime los caracteres impares de la tira con el juego desplazado hacia la izquierda, y coloca dos caracteres en cada bloque, resultando una extensión horizontal máxima de 64 caracteres. Los números empleados para indicar valores sobre los ejes, también precisan ser convertidos en tiras de caracteres de una forma consistente en cuanto a su longitud y/o precisión decimal. Este proceso se realiza mediante la subrutina "número", que se da a continuación en el listado 6.12. Recuérdese que la subrutina "variación de ancho" se empleó anteriormente para la realización de las etiquetas de la figura 6.6.

Listado 6.12

```
3000 REM grafica
3009 REM La rutina de simbolo es
 usada para marcar puntos en los
 graficos
3010 LET symbol=3500
3019 REM Calcula las escalas y d
ibuja los ejes
3020 INPUT "X VA DESDE ";XB;" HA
STA ";XT: IF XT<=XB THEN  GO TO
3020
3030 INPUT "Y VA DESDE ";YB;" HA
STA ";YT: IF YT<=YB THEN  GO TO
3030
```

```
3040 LET XSCALE=192/(XT-XB): LET
 YSCALE=128/(YT-YB)
3050 LET XO=INT (-XB*XSCALE+32.5
): LET YO=INT (-YB*YSCALE+24.5)
3059 REM Si el cero no esta en e
l rango mueve el eje al borde ap
ropiado
3060 IF YT<0 THEN  LET YO=153
3070 IF YB>0 THEN  LET YO=23
3080 IF XT<0 THEN  LET XO=224
3090 IF XB>0 THEN  LET XO=31
3100 PLOT XO,23: DRAW 0,128: PLO
T 31,YO: DRAW 192,0
3101 REM Usa la rutina de variac
ion de ancho para imprimir etiqu
etas en los ejes con 4 cifras
3110 LET XDIF=(XT-XB)/4: LET YDI
F=(YT-YB)/4
3120 LET X=XB: LET Y=YB: FOR J=1
 TO 5
3130 LET PX=INT ((X-XB)*XSCALE+3
2.5): LET PY=YO
3140 PLOT PX,PY-2: DRAW 0,4
3150 LET COL=INT (PX/8)-1: LET R
OW=INT ((175-PY)/8)+1
3160 LET A=X: GO SUB number: GO
SUB thin
3170 LET PY=INT ((Y-YB)*YSCALE+2
4.5): LET PX=XO
3180 PLOT PX-2,PY: DRAW 4,0
3190 LET COL=INT (PX/8)+1: LET R
OW=INT ((175-PY)/8)
3200 LET A=Y: GO SUB number: GO
SUB thin
3210 LET X=X+XDIF: LET Y=Y+YDIF:
 NEXT J
3220 INPUT "GRAFICO CONTINUO O D
ISCRETO ";D$: IF D$<>"c" AND D$<
>"d" THEN  GO TO 3220
3230 IF D$="d" THEN  GO TO 3320
3239 REM Lee la funcion a dibuja
r
3240 INPUT "F(x): y="; LINE F$
3250 LET X=XB: LET Y=VAL (F$): L
ET OY=INT ((Y-YB)*YSCALE+24.5)
```

```
3260 PLOT 32,OY
3270 FOR I=33 TO 224
3280 LET X=(I-32)/XSCALE+XB
3290 LET Y=VAL (F$): LET IY=INT
((Y-YB)*YSCALE+24.5)
3300 DRAW 1,IY-OY: LET OY=IY: NE
XT I
3310 RETURN
3319 REM Lee un grupo de puntos
para el grafico discreto
3320 INPUT "No. DE PUNTOS ";NP:
DIM X(NP): DIM Y(NP)
3330 FOR I=1 TO NP: INPUT ("X("+
STR$ I+") ");X(I);("  Y("+STR$ I
+") ");Y(I): NEXT I
3339 REM Ordena los puntos en or
den ascendente segun las X
3340 FOR I=1 TO NP-1: FOR J=I+1
TO NP
3350 IF X(J)<X(I) THEN  LET T=X(
I): LET X(I)=X(J): LET X(J)=T: L
ET T=Y(I): LET Y(I)=Y(J): LET Y(
J)=T
3360 NEXT J: NEXT I
3370 LET X=INT ((X(1)-XB)*XSCALE
+32.5): LET Y=INT ((Y(1)-YB)*YSC
ALE+24.5)
3380 PLOT X,Y: LET OX=X: LET OY=
Y: GO SUB symbol: PLOT OX,OY
3389 REM Une los puntos y situa
un simbolo en cada punto
3390 FOR I=2 TO NP
3400 LET X=INT ((X(I)-XB)*XSCALE
+32.5): LET Y=INT ((Y(I)-YB)*YSC
ALE+24.5)
3410 DRAW X-OX,Y-OY: LET OX=X: L
ET OY=Y: GO SUB symbol: PLOT OX,
OY
3420 NEXT I: RETURN

3500 REM simbolo
3510 DRAW 0,1: DRAW 1,0: DRAW 0,
-2: DRAW -2,0: DRAW 0,2
3520 RETURN
```

```
4700 REM numero
4710 LET A$=STR$ A: IF LEN A$<=4
 THEN  RETURN
4720 LET A$=A$( TO 4)
4730 RETURN
```

Ejercicio ***6.7***

Escriba una subrutina que realice la misma función que "número" y que le permita especificar el formato de la tira de caracteres que contenga un molde del formato numérico: por ejemplo la tira "# #.# # #" especificaría un número con dos dígitos en su parte entera y tres decimales.

Ejercicio ***6.8***

Construya los juegos de caracteres necesarios para las etiquetas numéricas (están incluidos en las cintas bajo los nombres "car3" y "car4"), empleados en el etiquetado de diagramas. La figura 6.8, que ha sido dibujada con el listado 5.2, le ayudará en su diseño.

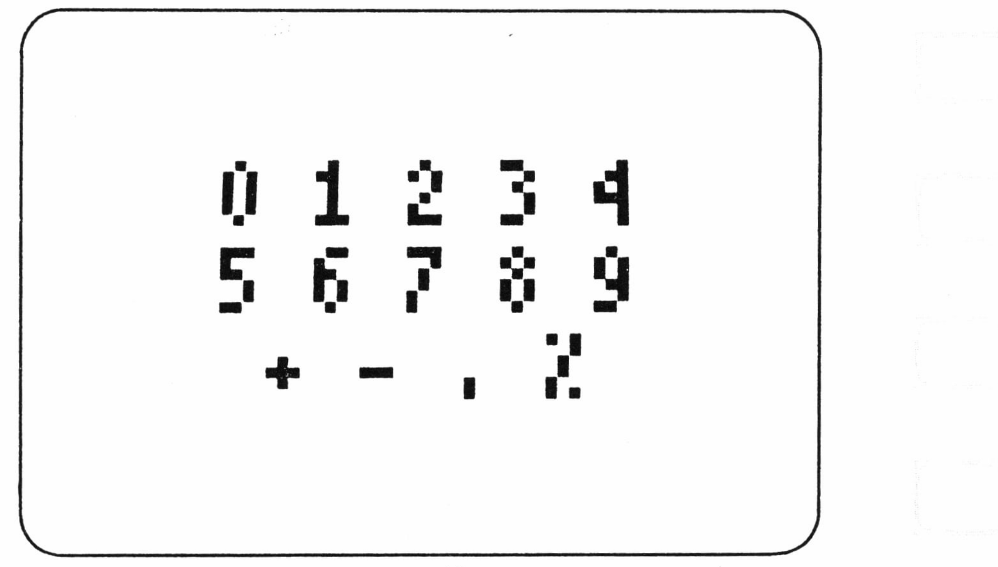


Figura 6.8

La elección que se presenta ahora es entre representar los puntos de una función continua, o dibujar un conjunto discreto de puntos que se unirán por líneas rectas tomando una forma de dientes de sierra. En la parte de la subrutina que

dibuja una función, calculamos la altura correspondiente a cada *pixel* sobre el eje x y se unen estos puntos con rectas. En la parte que dibuja un conjunto discreto de puntos, las coordenadas x e y de cada uno de los datos se ordenan en orden creciente de la coordenada x. Estos puntos se unen entonces con rectas. Damos un ejemplo de cada uno de los diagramas.

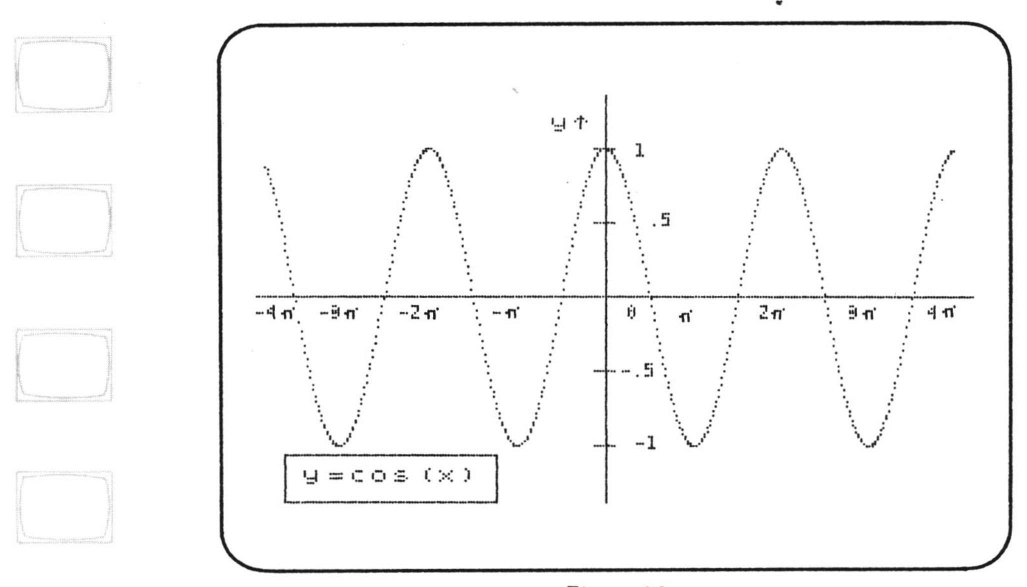


Figura 6.9

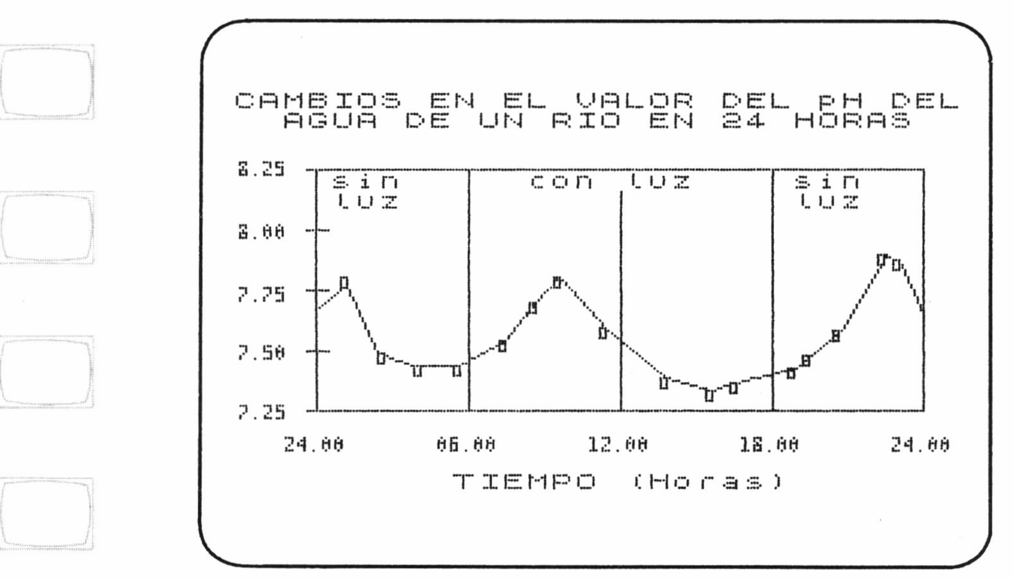


Figura 6.10

La figura 6.9 muestra la curva coseno típica, mientras que la 6.10 representa unos datos científicos discretos acerca del nivel del pH de un río.

Ejercicio 6.9

Puede observarse que el único dato que se precisa para una gráfica de este tipo es un conjunto de coordenadas en orden creciente de X, que luego se unirán con tramos rectos. Este conjunto de puntos puede crearse de dos maneras: bien por una sucesión de sentencias READ o por cálculo en una subrutina de varias líneas. En vez de eVALuar la función-tira de caracteres F$ podríamos escribir una subrutina que se empleará cada vez que necesitemos calcular un punto de la curva. Diseñe una subrutina que permita dibujar la gráfica de sen X/X, y evite el cálculo de sen 0/0.

Programas completos

Agrupamos los listados 6.6 ("programa principal", "carga caracteres del cassette", "set" y "opciones"), 6.1 ("rutina del cursor" y "grid"), 6.4 ("grabación" y "carga") bajo el nombre "rutdiag", y lo usaremos con 6.2 ("papel" y "tinta"), 6.3 ("punto" y "línea"), 6.5 ("etiquetado") y 6.7 ("crear caracteres" y "variación de ancho"), se encuentran todos en la cinta. Observe los cambios necesarios para la máquina de 16K que se mencionan en el apéndice A.

I. "rutdiag", 6.2, 6.3, 6.5, 6.7. Datos necesarios:

"DEFINIR CARACTERES?" (pulse) S para crear caracteres especiales girados y bloques para histogramas; en caso contrario, N.
"CARGAR CARACTERES?" S si se quiere cargar un juego de caracteres de cinta (inclusive los anteriores); N en caso contrario.
"CARGAR DIBUJO?" S para cargar un dibujo creado y almacenado en cinta. En caso contrario, N.
"DIBUJAR DIAGRAMA?" S para dibujar un histograma, círculo porcentual o gráfica. N si el diagrama sólo va a ser editado.
"ETIQUETAR DIBUJO?" S si se quiere usar "etiquetado". N en caso contrario.
"CAMBIAR COLOR PAPEL?" S o N. Si se introduce S, "COLOR?" (por ejemplo, 1: mueva el cursor; si es necesario, use "grid") y teclee el tamaño de la superficie en bloques FILAS*COLUMNAS (por ejemplo, 2*4).
"CAMBIAR COLOR TINTA?" Si S entonces "COLOR?" (por ejemplo, 5: mueva el cursor; si es necesario, use "grid") y teclee el tamaño de la superficie en bloques FILAS*COLUMNAS (por ejemplo, 3*1).
"DIBUJAR UN PUNTO?" Si S use el cursor para especificar el punto: "COLOR?" (por ejemplo, 6) y SOBREIMPRESION (1 ó 0).
"TERMINAR DIBUJO?" Si N se repite la secuencia de preguntas.

"SALVAR DIBUJO?" Si S, introduzca el nombre del dibujo. Experimente con las gráficas de datos.

II. "rutdiag" y el listado 6.8 ("histograma/tipo1"). Datos pedidos: por ejemplo,

"RANGO VERTICAL" 0 "A" 100
"N.º DE BARRAS" 6
"DATOS PARA LA BARRA 1" 56 "COLOR" 2
"DATOS PARA LA BARRA 2" 95 "COLOR" 6
"DATOS PARA LA BARRA 3" 20 "COLOR" 4
"DATOS PARA LA BARRA 4" 77 "COLOR" 5
"DATOS PARA LA BARRA 5" 54 "COLOR" 1
"DATOS PARA LA BARRA 6" 33 "COLOR" 3

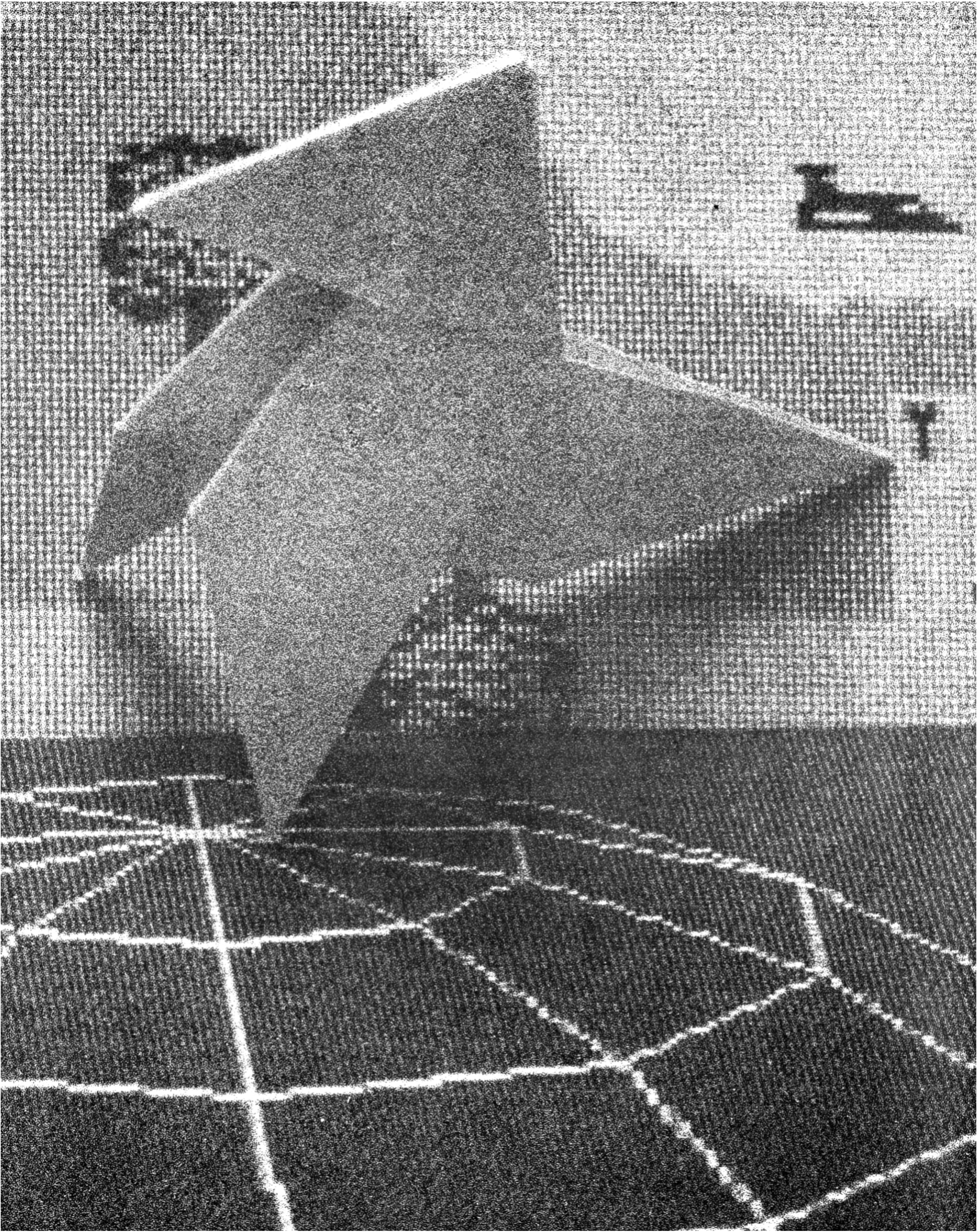
III. "rutdiag" y el listado 6.9 ("histograma/tipo2"). Datos necesarios: por ejemplo,

"RANGO VERTICAL" 0 "A" 50
"N.º DE BARRAS" 4
"COLOR PARA BARRA 1" 2
"MAX PARA BARRA 1" 44
"MIN PARA BARRA 1" 22
"COLOR PARA BARRA 2" 6
"MAX PARA BARRA 2" 36
"MIN PARA BARRA 2" 5
"COLOR PARA BARRA 3" 4
"MAX PARA BARRA 3" 42
"MIN PARA BARRA 3" 29
"COLOR PARA BARRA 4" 1
"MAX PARA BARRA 4" 31
"MIN PARA BARRA 4" 12

IV. "rutdiag" y listado 6.10 y 6.11 ("círculo porcentual", "sombreado", etc.). Datos requeridos: por ejemplo,

"N.º DE SEGMENTOS" 3
"COLOR" 0
"DATO 1" 1
"DATO 2" 2
"DATO 3" 3
Centre el círculo con el cursor, a continuación "RADIO (EN PIXELS)" 75
Centre la porción con el cursor. "SOMBREADO (x, y, xy, n)" xy: "INC." 8: "DESP." 5
Centre la porción con el cursor. "SOMBREADO" (x, y, xy, n)" y: "INC." 8: "DESP." 0
Centre la porción con el cursor. "SOMBREADO (x, y, xy, n)" n

V. "rutdiag" y el listado 6.12 ("gráfico", etc.). Las demandas del programa son autoexplicativas: como las anteriores.



7
Geometría cartesiana en tres dimensiones

Antes de comenzar el estudio de representaciones gráficas de objetos en el espacio tridimensional, es conveniente que demos un corto paseo por la geometría de coordenadas cartesianas en tres dimensiones. Al igual que sucedía en el espacio bidimensional, fijamos arbitrariamente un punto en el espacio, llamado *origen de coordenadas* (para abreviar origen). A continuación imaginamos tres rectas perpendiculares entre sí que se cortan en dicho punto; cada una de ellas alcanza hasta el infinito en ambas direcciones. Llamaremos a estas tres rectas el *eje x, eje y* y *eje z*. Cada eje, a su vez, tendrá una parte positiva y otra negativa, consideradas a partir del origen: es decir, las distancias medidas desde el origen a lo largo de un eje serán positivas en una parte del mismo y negativas en la otra. Para simplificar, podemos imaginar que los ejes x e y son equivalentes a los que dibujábamos en el espacio de dos dimensiones, ambos sobre la hoja de este libro, por ejemplo. Como antes, colocaremos el eje x "horizontal" y con la parte positiva hacia la derecha del origen, y el eje y "vertical", y con su parte positiva hacia la parte superior del libro. Una vez fijadas las posiciones de estos dos ejes, la posición del eje z queda predeterminada: debe ser perpendicular a la página (ya que es perpendicular tanto al eje x como al eje y). Nos queda ahora escoger la parte positiva del eje z. La parte positiva puede entrar en la página *(convenio de ejes de mano izquierda)* o salir de la misma *(convenio de la mano derecha). En este libro utilizaremos siempre el convenio de mano izquierda.* Añadiremos que cualquier operación realizada con ejes "zurdos" es equivalente si se escogiese el otro convenio, con tal que uno sea consistente con la notación elegida.

Un punto $\boldsymbol{p}$ en el espacio queda definido por un trío de coordenadas o vector (X, Y, Z), donde los valores individuales de las coordenadas representan proyecciones perpendiculares del punto a los respectivos ejes x, y, z. Llamamos proyección al

punto del eje tal que si unimos dicho punto con el punto $\boldsymbol{p}$ el segmento obtenido es perpendicular al eje.

Vamos a definir dos operaciones para vectores tridimensionales. Supongamos que tenemos dos vectores $\boldsymbol{p}_1 \equiv (x_1, y_1, z_1)$ y $\boldsymbol{p}_2 \equiv (x_2, y_2, z_2)$.

Se llama *multiplicación por un escalar* $k\boldsymbol{p}_1 \equiv (k \times x_1, k \times y_1, k \times z_1)$ al resultado de multiplicar las tres coordenadas por un número k.

Suma de vectores $\boldsymbol{p}_1 + \boldsymbol{p}_2 \equiv (x_1 + x_2, y_1 + y_2, z_1 + z_2)$ es la operación por la cual se suman las coordenadas de dos vectores por separado, es decir, las coordenadas x por un lado, las y por otro y las z por otro. El resultado es un nuevo vector.

Definición de un línea recta

El siguiente objeto que definiremos en el espacio tridimensional es una *línea recta* que pase por dos puntos, $\boldsymbol{p}_1 \equiv (x_1, y_1, z_1)$ y $\boldsymbol{p}_2 \equiv (x_2, y_2, z_2)$. Describiremos las coordenadas de un punto cualquiera $\boldsymbol{p} \equiv (x, y, z)$ en la recta por medio de tres ecuaciones:

$$(x - x_1) \times (y_2 - y_1) = (y - y_1) \times (x_2 - x_1)$$

$$(y - y_1) \times (z_2 - z_1) = (z - z_1) \times (y_2 - y_1)$$

$$(z - z_1) \times (x_2 - x_1) = (x - x_1) \times (z_2 - z_1)$$

Aunque el sistema propuesto tiene tres ecuaciones y tres incógnitas, se puede demostrar que están interrelacionadas (lo que se llama ecuaciones *linealmente dependientes*) y que, por tanto, no tienen una solución única. Es lógico que sea así, ya que intentamos describir un punto cualquiera de la línea, no un punto determinado. Con estas ecuaciones podemos calcular dos coordenadas en función de la tercera (véase ejemplo 7.1).

Tal como sucedía en dos dimensiones, una recta se puede representar de varias maneras diferentes; de hecho, introduciremos ahora una segunda forma que probablemente es más útil. Un punto cualquiera de la recta se representa como un vector que depende de un único número real μ, que se calcula como vector suma de dos vectores multiplicados por escalares.

$$\boldsymbol{p}(\mu) \equiv (1 - \mu)\boldsymbol{p}_1 + \mu\boldsymbol{p}_2 \quad \text{donde} \quad -\infty < \mu < \infty$$

es decir,

$$\boldsymbol{p}(\mu) \equiv ((1 - \mu) \times x_1 + \mu \times x_2, (1 - \mu) \times y_1 + \mu \times y_2, (1 - \mu) \times z_1 + \mu \times z_2)$$

Esta forma de representación equivale exactamente a la forma paramétrica en dos dimensiones que utilizábamos en el capítulo 3 para definir una recta. El símbolo μ que aparece entre paréntesis detrás de $\boldsymbol{p}$ indica que $\boldsymbol{p}$ depende de μ; sin embargo, una vez conocido este hecho (μ) puede ignorarse. Obsérvese que cuando $\mu = 0$, la ecuación da el punto $\boldsymbol{p}_1$, y si $\mu = 1$, resulta $\boldsymbol{p}_2$.

Podemos reorganizar esta expresión de la forma

$$\boldsymbol{p}(\mu) \equiv \boldsymbol{p}_1 + \mu(\boldsymbol{p}_2 - \boldsymbol{p}_1)$$

Al igual que en su homólogo bidimensional, $\boldsymbol{p}_1$ se llama *vector de base* y $(\boldsymbol{p}_2 - \boldsymbol{p}_1)$ *vector de dirección.* También observamos aquí la doble interpretación que se puede asignar a un vector. En efecto, un vector puede utilizarse para identificar unívocamente un punto en el espacio tridimensional, o bien puede establecer una dirección: en concreto, cualquier paralela a la recta que una el origen al vector (considerado como un punto). Podemos movernos por la recta en uno u otro sentido, de manera que definiremos un *sentido positivo* cuando viajemos desde el origen hacia el punto, y un *sentido negativo* cuando vayamos desde el punto hacia el origen. De esta forma, los vectores $\boldsymbol{d} \equiv (x, y, z)$ y $-\boldsymbol{d} \equiv (-x, -y, -z)$ representan la misma recta en el espacio pero definen dos sentidos contrarios. Se define la longitud de un vector $\boldsymbol{d} \equiv (x, y, z)$ (también llamada su módulo o valor absoluto) como $|\boldsymbol{d}|$, distancia desde el extremo del vector al origen

$$|\boldsymbol{d}| = \sqrt{(x^2 + y^2 + z^2)}$$

Podemos localizar cualquier punto de la recta $\boldsymbol{p} + \mu\boldsymbol{d}$ colocándonos en el punto $\boldsymbol{p}$ y moviéndonos a lo largo de una línea paralela a la dirección $\boldsymbol{d}$ una distancia $\mu|\boldsymbol{d}|$ en sentido positivo de $\boldsymbol{d}$ si μ es positivo, y en sentido negativo en caso contrario. Obsérvese que se puede utilizar cualquier punto de la recta como vector de base, y que el vector de dirección se puede sustituir por cualquier múltiplo del mismo distinto de cero.

El vector de dirección $\boldsymbol{d} \equiv (x, y, z)$ forma ángulos θ_x, θ_y y θ_z con las direcciones positivas de los ejes x, y, z respectivamente; tenemos, por tanto, las relaciones

$$x:y:z = \cos\theta_x : \cos\theta_y : \cos\theta_z$$

lo que significa que $\boldsymbol{d} \equiv (\lambda \times \cos\theta_x, \lambda \times \cos\theta_y, \lambda \times \cos\theta_z)$ para algún λ.

Por las propiedades de la geometría en tres dimensiones sabemos que

$$\cos^2\theta_x + \cos^2\theta_y + \cos^2\theta_z = 1$$

Así pues, si $\lambda = |\boldsymbol{d}|$, y además el vector de dirección tiene módulo unidad (es decir, módulo $= \lambda = 1$), las coordenadas de dicho vector serán $(\cos\theta_x, \cos\theta_y, \cos\theta_z)$; o lo que es lo mismo, $\lambda = 1$. Cuando las coordenadas de un vector de dirección se dan de esta forma, se denominan *cosenos directores* del conjunto de líneas definido por el vector. En general, si el vector de dirección es $\boldsymbol{d} \equiv (x, y, z)$, los cosenos directores son

$$\left(\frac{x}{|\boldsymbol{d}|}, \frac{y}{|\boldsymbol{d}|}, \frac{z}{|\boldsymbol{d}|}\right)$$

Ejemplo 7.1

Describa la recta que pasa por los puntos $(1, 2, 3)$ y $(-1, 0, 2)$, usando los tres métodos vistos hasta ahora.

Un punto cualquiera (x, y, z) de la recta satisface las ecuaciones

$$(x - 1) \times (0 - 2) = (y - 2) \times (-1 - 1) \quad \text{es decir,} \quad -2x \pm 2y = 2 \tag{7.1}$$

$$(y - 2) \times (2 - 3) = (z - 3) \times (0 - 2) \quad\quad -y + 2z = 4 \tag{7.2}$$

y

$$(z - 3) \times (-1 - 1) = (x - 1) \times (2 - 3) \quad\quad -2z + x = -5 \tag{7.3}$$

Se observa que la ecuación 7.1 es igual a la suma de las ecuaciones 7.2 y 7.3 multiplicada por -2. Consideraremos, por consiguiente, las dos últimas ecuaciones únicamente, obteniendo

$$y = 2z - 4 \text{ y } x = 2z - 5$$

así, un punto cualquiera de la recta depende de una única variable, en este caso z, y está dado por $(2z - 5, 2z - 4, z)$. El resultado se puede comprobar con facilidad sustituyendo $z = 3$, con lo que obtenemos $(1, 2, 3)$, y sustituyendo $z = 2$ obtenemos $(-1, 0, 2)$, que eran los dos puntos originales que definían la recta.

Expresándolo en forma vectorial, un punto cualquiera de la recta (dependiendo de μ) será

$$\boldsymbol{p}(\mu) \equiv (1 - \mu)(1, 2, 3) + \mu(-1, 0, 2) \equiv (1 - 2\mu, 2 - 2\mu, 3 - \mu)$$

Una vez más las coordenadas dependen de una variable únicamente (μ), y podemos comprobar la validez de la ecuación propuesta haciendo $\boldsymbol{p}(0) \equiv (1, 2, 3)$ y $\boldsymbol{p}(1) \equiv (-1, 0, 2)$.

La tercera forma de representación se basa en un vector de base y en un vector de dirección:

$$\boldsymbol{p}(\mu) \equiv (1, 2, 3) + \mu(-2, -2, -1)$$

donde $(1, 2, 3)$ actúa de vector de base y $(-2, -2, -1)$ de dirección. (Obsérvese que el módulo es $\sqrt{(4 + 4 + 1)} = \sqrt{9} = 3$.) Observamos también que cualquier punto de la línea puede actuar como vector de base, de manera que podemos adoptar otro punto $\boldsymbol{p}'$ obteniendo

$$\boldsymbol{p}'(\mu) \equiv (-1, 0, 2) + \mu(-2, -2, -1)$$

Si expresamos el vector de dirección en forma de cosenos directores $(-2/3, -2/3, -1/3)$ podemos representar la recta en una versión distinta de la misma forma vector base/vector de dirección.

$$\boldsymbol{p}''(\mu) \equiv (1, 2, 3) + \mu(-2/3, -2/3, -1/3)$$

Evidentemente, el mismo valor de μ dará distintos puntos para representaciones diferentes de la recta; por ejemplo, $\boldsymbol{p}(3) \equiv (-5, -4, 0)$, $\boldsymbol{p}'(3) \equiv (-7, -6, -1)$ y $\boldsymbol{p}''(3) \equiv$ $\equiv (-1, 0, 2)$. Los ángulos formados por esta recta y los ejes son 131,81 grandos $= \cos^{-1}$ $(-2/3)$, 131,81 grados y 109,47 grados $= \cos^{-1}(-1/3)$ respectivamente, al eje x, eje y, eje z.

Angulo formado por dos vectores de dirección

El cálculo de este ángulo requiere que previamente definamos el $\cdot$, llamado *producto escalar*. Este operador actúa sobre dos vectores y su resultado es un número real. Así,

$$\boldsymbol{p} \cdot \boldsymbol{q} = (x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1 \times x_2 + y_1 \times y_2 + z_1 \times z_2$$

Si $\boldsymbol{p}$ y $\boldsymbol{q}$ son vectores unitarios (es decir, están expresados en forma de cosenos directores) y θ es el ángulo entre las rectas, se cumple $\cos\theta = \boldsymbol{p} \cdot \boldsymbol{q}$ (véase la expresión equivalente en dos dimensiones mostrada en el capítulo 3). Por consiguiente, en general el ángulo entre dos vectores de dirección $\boldsymbol{p}$ y $\boldsymbol{q}$ (que suponemos se cortan en el origen) será

$$\cos^{-1}\left(\frac{\boldsymbol{p}}{|\boldsymbol{p}|} \cdot \frac{\boldsymbol{q}}{|\boldsymbol{q}|}\right)$$

Evidentemente, $\boldsymbol{p}$ y $\boldsymbol{q}$ serán perpendiculares si y sólo si $\boldsymbol{p} \cdot \boldsymbol{q} = 0$.

Definición de un plano

A continuación definiremos un plano en el espacio tridimensional. Un punto cualquiera del plano $\boldsymbol{x} \equiv (x, y, z)$ viene dado por la ecuación vectorial

$$\boldsymbol{n} \cdot \boldsymbol{x} = k$$

donde k es un número, y $\boldsymbol{n}$ el vector de dirección del conjunto de líneas que son perpendiculares *(normales)* al plano (véase ejemplo 7.2). Si $\boldsymbol{a}$ es un punto cualquiera del plano, se cumplirá $\boldsymbol{n} \cdot \boldsymbol{a} = k$, por lo que, sustituyendo k en la ecuación anterior, obtenemos

$$\boldsymbol{n} \cdot \boldsymbol{x} = \boldsymbol{n} \cdot \boldsymbol{a} \quad \text{o bien} \quad \boldsymbol{n} \cdot (\boldsymbol{x} - \boldsymbol{a}) = 0$$

La última ecuación es evidente y recordamos la propiedad ya mencionada del producto escalar que hace que dos rectas perpendiculares tengan un producto nulo. Para cualquier punto $\boldsymbol{x} \equiv (x, y, z)$ en el plano distinto de $\boldsymbol{a}$, sabemos que $(\boldsymbol{x} - \boldsymbol{s})$ puede considerarse como la dirección de una recta en el plano. Al ser $\boldsymbol{n}$ normal

al plano, y por tanto perpendicular a cualquier línea de dicho plano, el producto $\boldsymbol{n} \cdot (\boldsymbol{x} - \boldsymbol{a}) = \cos(\pi/2) = 0$.

Expandiendo la ecuación original del plano con una normal $\boldsymbol{n} \equiv (n_1, n_2, n_3)$, obtenemos la forma general de representación por coordenadas de un plano:

$$(n_1, n_2, n_3) \cdot (x, y, z) = n_1 \times x + n_2 \times y + n_3 \times z = k$$

Obsérvese que dos planos cuyas normales sean $\boldsymbol{m}$ y $\boldsymbol{n}$ serán paralelos si y sólo si una de las normales es múltiplo de la otra, es decir, $\boldsymbol{n} = \lambda \boldsymbol{m}$ para algún λ.

Punto de intersección de una recta y un plano

Supongamos la recta definida por $\boldsymbol{b} + \mu \boldsymbol{d}$ y el plano definido por $\boldsymbol{n} \cdot \boldsymbol{x} = k$. Al ser el punto de intersección común a la recta y al plano, tendremos que encontrar el único valor de μ (si existe) para el cual

$$\boldsymbol{n} \cdot (\boldsymbol{b} + \mu \boldsymbol{d}) = k$$

es decir, $\mu = (k - \boldsymbol{n} \cdot \boldsymbol{b})/(\boldsymbol{n} \cdot \boldsymbol{d})$ siempre que $\boldsymbol{n} \cdot \boldsymbol{d} \neq 0$.

Cuando la recta y el plano son paralelos se cumplirá $\boldsymbol{n} \cdot \boldsymbol{d} = 0$, y no habrá puntos de intersección.

Distancia de un punto a un plano

La distancia de un punto $\boldsymbol{p}_1$ al plano $\boldsymbol{n} \cdot \boldsymbol{x} = k$ es la distancia del punto $\boldsymbol{p}_1$ al punto $\boldsymbol{p}_2$ situado en el plano que sea más próximo a $\boldsymbol{p}_1$. Si trazamos una perpendicular al plano por el punto $\boldsymbol{p}_2$, dicha perpendicular deberá por consiguiente pasar por $\boldsymbol{p}_1$. La ecuación de esta recta puede expresarse $\boldsymbol{p}_1 + \mu \boldsymbol{n}$ y el valor de μ que define $\boldsymbol{p}_2$ debe ser tal que

$$\mu = (k - \boldsymbol{n} \cdot \boldsymbol{p}_1)/(\boldsymbol{n} \cdot \boldsymbol{n})$$

de acuerdo con la ecuación anterior, la distancia del punto $\boldsymbol{p}_2 \equiv \boldsymbol{p}_1 + \mu \boldsymbol{n}$ al punto $\boldsymbol{p}_1$ será

$$\mu \times |\boldsymbol{n}| = |k - \boldsymbol{n} \cdot \boldsymbol{p}_1|/|\boldsymbol{n}|$$

Como caso particular, si $\boldsymbol{p}_1$ es el origen de coordenadas $\boldsymbol{O}$, la distancia del plano al origen será $|k|/|\boldsymbol{n}|$. Además, si $\boldsymbol{n}$ es un coseno director, la distancia al origen estará expresada por $|k|$, valor absoluto del número real k.

Ejemplo 7.2

Encuéntrese el punto de intersección de la recta que une (1, 2, 3) y (−1, 0, 2) con el plano $(0, -2, 1) \cdot \boldsymbol{x} = 5$; calcúlese también la distancia del plano al origen.

$$\boldsymbol{b} \equiv (1, 2, 3)$$

$$\boldsymbol{n} \equiv (0, -2, 1)$$

$$\boldsymbol{d} \equiv (-1, 0, 2) - (1, 2, 3) \equiv (-2, -2, -1)$$

$$\boldsymbol{n} \cdot \boldsymbol{b} = (0 \times 1 + -2 \times 2 + 1 \times 3) = -1$$

$$\boldsymbol{n} \cdot \boldsymbol{d} = (0 \times -2 + -2 \times -2 + 1 \times -1) = 3$$

por tanto el valor de μ del punto de intersección será $(5 - (-1))/3 = 2$, y el vector del punto es

$$(1, 2, 3) + 2(-2, -2, -1) \equiv (-3, -2, 1)$$

la distancia del punto al origen será $5/|\boldsymbol{n}| = 5/\sqrt{5} = \sqrt{5}$.

El programa del listado 7.1 nos permite calcular el punto de intersección (vector P) de una recta y un plano. La recta tiene un vector de base B y una dirección D, y el plano tiene una normal N y una constante K. obsérvese que trabajamos con números decimales, y estamos por tanto sujetos a errores de redondeo; por tanto no podemos comprobar si el producto escalar es exactamente cero. Lo único que podemos hacer es considerar que es lo suficientemente pequeño como para resultar despreciable; se deja al buen criterio del programador la definición de "lo suficientemente pequeño" (en el Spectrum, la sexta cifra decimal es bastante razonable).

Listado 7.1

```
 100 REM interseccion de recta y
 plano
 110 DIM B(3): DIM D(3): DIM N(3
): DIM P(3): DIM A$(8)
 120 INPUT "PUNTO DE LA RECTA ",
"(";B(1);",";B(2);",";B(3);")"
 130 PRINT AT 1,0;"PUNTO DE LA R
ECTA","(";B(1);",";B(2);",";B(3)
;")"
 140 INPUT "VECTOR DIRECTOR DE L
A RECTA","(";D(1);",";D(2);",";D
(3);")"
 150 PRINT AT 4,0;"VECTOR DIRECT
```

```
OR DE LA RECTA","(";D(1);",";D(2
);",";D(3);")"
 160 INPUT "VECTOR NORMAL AL PLA
NO","(";N(1);",";N(2);",";N(3);"
)"
 170 PRINT AT 7,0;"VECTOR NORMAL
 AL PLANO","(";N(1);",";N(2);","
;N(3);")"
 180 INPUT "CONSTANTE DEL PLANO
";K
 188 REM Calcula el punto de int
erseccion (P(1),P(2),P(3))
 189 REM de la recta y el plano;
 datos de entrada arriba
 190 PRINT AT 10,0;"CONSTANTE DE
L PLANO ";K
 200 LET DOT=N(1)*D(1)+N(2)*D(2)
+N(3)*D(3)
 209 REM Producto escalar=0; no
hay interseccion
 210 IF ABS DOT<0.000001 THEN  P
RINT AT 15,0;"NO HAY PUNTO DE IN
TERSECCION": STOP
 220 LET MU=(K-N(1)*B(1)-N(2)*B(
2)-N(3)*B(3))/DOT
 230 FOR I=1 TO 3: LET P(I)=B(I)
+MU*D(I): IF ABS P(I)<0.000001 T
HEN  LET P(I)=0
 240 NEXT I: PRINT AT 15,0;"PUNT
O DE INTERSECCION ","(";
 249 REM Salida de datos
 250 FOR I=1 TO 3: LET A$=STR$ P
(I): FOR J=1 TO 8
 260 IF A$(J)<>" " THEN  PRINT A
$(J);
 270 NEXT J: IF I<>3 THEN  PRINT
 ",";
280 NEXT I: PRINT ")"
290 STOP
```

Punto de intersección entre dos rectas

Supongamos que tenemos dos rectas definidas como $\boldsymbol{b}_1 + \mu \boldsymbol{d}_1$ y $\boldsymbol{b}_2 + \lambda \boldsymbol{d}_2$. Para que dichas rectas tengan un punto de intersección, se deberá cumplir que no sean paralelas y además que estén en el mismo plano. En estas condiciones, su punto de

intersección se calcula encontrando los valores de μ y λ que satisfacen la ecuación vectorial (equivalente a tres ecuaciones de coordenadas separadas)

$$\boldsymbol{b}_1 + \mu \boldsymbol{d}_1 = \boldsymbol{b}_2 + \lambda \boldsymbol{d}_2$$

Tenemos, por consiguiente, tres ecuaciones con dos incógnitas; esto significa que para que las ecuaciones tengan sentido, deberá existir un par de ecuaciones independientes, y la tercera será una combinación lineal de las otras dos. Dos rectas son paralelas cuando el vector de dirección de una de ellas es un múltiplo del otro. Por consiguiente escogeremos dos ecuaciones independientes, buscaremos los valores de μ y λ (tenemos ahora dos ecuaciones con dos incógnitas), y las sustituiremos en la tercera ecuación para comprobar si son consistentes. En el ejemplo 7.3, más adelante, se demuestra este método; el listado 7.2 es una forma de implementarlo en un ordenador. La primera recta tiene su base y dirección almacenadas en los vectores B y D, y la segunda en C y E: el punto de intersección calculado se almacena el vector P.

Obsérvese que si las dos ecuaciones independientes son

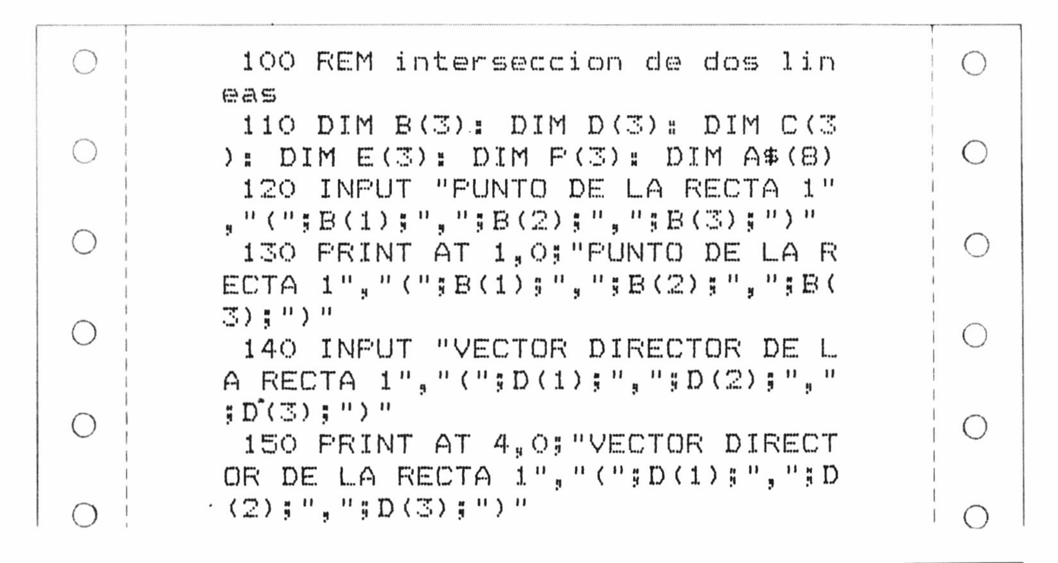
$$a_{11} \times \mu + a_{12} \times \lambda = b_1$$

$$a_{21} \times \mu + a_{22} \times \lambda = b_2$$

se ha de cumplir que el *determinante* $\Delta = a_{11} \times a_{22} - a_{12} \times a_{21}$ de este par de ecuaciones ha de ser distinto de cero (ya que las ecuaciones son independientes); obtendremos las soluciones

$$\mu = (a_{22} \times b_1 - a_{12} \times b_2)/\Delta \quad \text{y} \quad \lambda = (a_{11} \times b_2 - a_{21} \times b_1)/\Delta$$

Listado 7.2



```
 100 REM interseccion de dos lin
eas
 110 DIM B(3): DIM D(3): DIM C(3
): DIM E(3): DIM P(3): DIM A$(8)
 120 INPUT "PUNTO DE LA RECTA 1"
,"(";B(1);",";B(2);",";B(3);")"
 130 PRINT AT 1,0;"PUNTO DE LA R
ECTA 1","(";B(1);",";B(2);",";B(
3);")"
 140 INPUT "VECTOR DIRECTOR DE L
A RECTA 1","(";D(1);",";D(2);","
;D(3);")"
 150 PRINT AT 4,0;"VECTOR DIRECT
OR DE LA RECTA 1","(";D(1);",";D
(2);",";D(3);")"
```

```
 160 INPUT "PUNTO DE LA RECTA 2"
,"(";C(1);",";C(2);",";C(3);")"
 170 PRINT AT 7,0;"PUNTO DE LA R
ECTA 2","(";C(1);",";C(2);",";C(
3);")"
 180 INPUT "VECTOR DIRECTOR DE L
A RECTA 2","(";E(1);",";E(2);","
;E(3);")"
 190 PRINT AT 10,0;"VECTOR DIREC
TOR DE LA RECTA 2","(";E(1);",";
E(2);",";E(3);")"
 198 REM Calcula el punto de int
erseccion (P(1),P(2),P(3))
 199 REM de dos lineas, datos de
 entrada arriba
 200 FOR I=1 TO 3
 210 LET J=I+1: IF J=4 THEN  LET
 J=1
 220 LET DELTA=E(I)*D(J)-E(J)*D(
I)
 230 IF ABS DELTA>0.000001 THEN
 GO TO 260
 240 NEXT I
 249 REM No se pueden encontrar
2 ecuaciones independientes; las
 rectas no se cortan
 250 PRINT AT 15,0;"LAS RECTAS N
O SE CORTAN": STOP
 259 REM Calcula los valores de
MU y LANDA del punto de intersec
cion
 260 LET MU=(E(I)*(C(J)-B(J))-E(
J)*(C(I)-B(I)))/DELTA
 270 LET LAMBDA=(D(I)*(C(J)-B(J)
)-D(J)*(C(I)-B(I)))/DELTA
 279 REM No hay solucion si MU y
 LANDA no satisfacen la tercera
ecuacion
 280 LET K=J+1: IF K=4 THEN  LET
 K=1
 290 IF ABS (B(K)+MU*D(K)-C(K)-L
AMBDA*E(K))>0.000001 THEN  GO TO
 250
 299 REM Calcula (P(1),P(2),P(3)
) usando el valor de MU
```

```
 300 FOR I=1 TO 3: LET P(I)=B(I)
+MU*D(I): IF ABS P(I)<0.000001 T
HEN LET P(I)=0
 310 NEXT I: PRINT AT 15,0;"PUNT
O DE INTERSECCION ","(";
 319 REM Salida de datos
 320 FOR I=1 TO 3: LET A$=STR$ P
(I): FOR J=1 TO 8
 330 IF A$(J)<>" " THEN PRINT A
$(J);
 340 NEXT J: IF I<>3 THEN PRINT
",";
 350 NEXT I: PRINT ")"
 360 STOP
```

Ejemplo 7.3

Encuéntrese el punto de intersección (si existe) de

a) $(1, 1, 1) + \mu(2, 1, 3)$ con $(0, 0, 1) + \lambda(-1, 1, 1)$,

b) $(2, 3, 4) + \mu(1, 1, 1)$ con $(-2, -3, -4) + \lambda(1, 2, 3)$.

De *a*) obtenemos las tres ecuaciones:

$$1 + 2\mu = 0 - \lambda \tag{7.4}$$

$$1 + \mu = 0 + \lambda \tag{7.5}$$

$$1 + 3\mu = 1 + \lambda \tag{7.6}$$

A partir de 7.4 y 7.5 obtenemos $\mu = -2/3$ y $\lambda = 1/3$ que sustituimos en la ecuación 7.6 La parte izquierda de esta ecuación da $1 + 3 \times (-2/3) = -1$ mientras que la parte derecha resulta $1 + 1 \times (1/3) = 4/3$. Al ser diferentes los resultados, las dos rectas, obviamente, no se cortan. A partir de *b*) tenemos las ecuaciones

$$2 + \mu = -2 + \lambda \tag{7.7}$$

$$3 + \mu = -3 + 2\lambda \tag{7.8}$$

$$4 + \mu = -4 + 3\lambda \tag{7.9}$$

y de 7.7 y 7.8 obtenemos $\mu = -2$ y $\lambda = 2$; sustituyendo estos valores en la ecuación 7.9 comprobamos que la parte izquierda de la ecuación es igual a la parte derecha = 2). Por consiguiente el punto de intersección es

$$(2, 3, 4) + -2(1, 1, 1) = (-2, -3, -4) + 2(1, 2, 3) = (0, 1, 2)$$

Plano definido por tres puntos no colineales

Para resolver este problema necesitamos introducir un nuevo operador vectorial, X , producto vectorial, que actúa sobre dos vectores ***p*** y ***q*** dando como resultado otro vector:

$$\boldsymbol{p} \times \boldsymbol{q} = (p_1, p_2, p_3) \times (q_1, q_2, q_3)$$
$$= (p_2 \times q_3 - p_3 \times q_2, p_3 \times q_1 - p_1 \times q_3, p_1 \times q_2 - p_2 \times q_1)$$

Si ***p*** y ***q*** son dos vectores de dirección no paralelos, el producto vectorial ***p*** X ***q*** será el vector de dirección perpendicular a ambos, ***p*** y ***q***. Es importante destacar que esta operación *no es conmutativa.* Es decir, en general para unos valores dados de ***p*** y ***q***, $\boldsymbol{p} \times \boldsymbol{q} \neq \boldsymbol{q} \times \boldsymbol{p}$. Estos dos productos vectoriales representan vectores de la misma dirección sobre la misma línea pero en sentidos opuestos. Por ejemplo, (1, 0, 0) X (0, 1, 0) = (0, 0, 1) pero (0, 1, 0) X (1, 0, 0) = (0, 0, −1); (0, 0, 1) y (0, 0, −1) son paralelos al eje *z* (y por consiguiente perpendiculares a las direcciones (1, 0, 0) y (0, 1, 0)), pero tienen sentidos opuestos. El listado 7.3 es un programa principal que llama a las subrutinas “producto vectorial” (“vecprod”) y “producto escalar” (“dotprod”); la primera calcula el producto vectorial entre dos vectores L y M y da como resultado el vector N; la segunda calcula el producto escalar de los vectores L y M); estas dos subrutinas se dan en el listado 7.4.

Listado 7.3

```
 100 REM producto escalar y vect
orial
 110 LET vecprod=300: LET dotpro
d=400
 120 DIM L(3): DIM M(3): DIM N(3
)
 130 INPUT "VECTOR L","(";L(1);"
,";L(2);",";L(3);")"
 140 PRINT AT 1,0;"VECTOR L","("
;L(1);",";L(2);",";L(3);")"
 150 INPUT "VECTOR M","(";M(1);"
,";M(2);",";M(3);")"
 160 PRINT AT 4,0;"VECTOR M","("
;M(1);",";M(2);",";M(3);")"
 170 GO SUB vecprod
 180 PRINT AT 8,0;"PRODUCTO VECT
ORIAL","(";N(1);",";N(2);",";N(3
);")"
 190 GO SUB dotprod
```

```
 200 PRINT AT 11,0;"PRODUCTO ESC
ALAR",DOT
 210 STOP
```

Listado 7.4

```
 300 REM producto vectorial

 301 REM Datos de entrada L(3),M
(3)
 302 REM Datos de salida N(3)
 309 REM N es el producto vector
ial de L y M
 310 LET NI=2: LET NNI=3
 320 FOR I=1 TO 3
 330 LET N(I)=L(NI)*M(NNI)-L(NNI
)*M(NI)
 340 LET NI=NNI: LET NNI=NI+1: I
F NNI=4 THEN  LET NNI=1
 350 NEXT I
 360 RETURN

 400 REM producto escalar
 401 REM Datos de entrada L(3),M
(3)
 402 REM Datos de salida DOT
 409 REM DOT es el producto esca
lar de L y M
 410 LET DOT=0
 420 FOR I=1 TO 3: LET DOT=DOT+L
(I)*M(I): NEXT I
 430 RETURN
```

Supongamos que tenemos tres puntos $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, no colineales (es decir, que no forman una línea recta). En esas condiciones, los dos vectores $\mathbf{p}_2 - \mathbf{p}_1$ y $\mathbf{p}_3 - \mathbf{p}_1$ representan las direcciones de dos rectas que coinciden en el punto $\mathbf{p}_1$, estando ambos situados en el plano que contiene a los tres puntos. Sabemos que la normal a dicho plano es perpendicular a cada línea del plano, por lo que lo será a las dos líneas mencionadas. Además, al ser los puntos no colineales, se cumplirá que $\mathbf{p}_2 - \mathbf{p}_1 \neq$ $\neq \mathbf{p}_3 - \mathbf{p}_1$, y la normal al plano es $(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$; como $\mathbf{p}_1$ está situada en el plano la ecuación es

$$((\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p} - \mathbf{p}_1)) \cdot (\mathbf{x} - \mathbf{p}_1) = 0$$

Ejemplo 7.4

Calcular la ecuación de coordenadas del plano que contiene a los puntos (0, 1, 1), (1, 2, 3) y (−2, 3, −1).

Se calcula por medio de un punto cualquiera $x \equiv (x, y, z)$ donde

$$(((1, 2, 3) - (0, 1, 1)) \times ((-2, 3, -1) - (0, 1, 1))) \cdot ((x, y, z) - (0, 1, 1)) = 0$$

es decir,

$$((1, 1, 2) \times (-2, 2, -2) \cdot (x, y - 1, z - 1) = 0$$

o bien

$$(-6, -2, 4) \cdot (x, y - 1, z - 1) = 0$$

que expresada en forma de coordenadas es $-6x - 2y + 4z - 2 = 0$ o en su forma equivalente $3x + y - 2z = -1$.

Punto de intersección de tres planos

Suponemos que los tres planos están definidos por las ecuaciones 7.10 a 7.12 mostradas a continuación. El punto de intersección de estos tres planos, $\boldsymbol{x} \equiv (x, y, z)$, debe pertener a los tres planos y satisfacer el sistema

$$\boldsymbol{n}_1 \cdot \boldsymbol{x} = k_1 \tag{7.10}$$

$$\boldsymbol{n}_2 \cdot \boldsymbol{x} = k_2 \tag{7.11}$$

$$\boldsymbol{n}_3 \cdot \boldsymbol{x} = k_3 \tag{7.12}$$

donde $\boldsymbol{n}_1 \equiv (n_{11}, n_{12}, n_{13})$, $\boldsymbol{n}_2 \equiv (n_{21}, n_{22}, n_{23})$ y $\boldsymbol{n}_3 \equiv (n_{31}, n_{32}, n_{33})$. Podemos reescribir estas tres ecuaciones en forma de una ecuación matricial

$$\begin{pmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}$$

cuya solución para $\boldsymbol{x}$ viene dada por el *vector columna*

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{pmatrix}^{-1} \times \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}$$

De este modo, cualquier cálculo de la intersección de tres planos implica necesariamente calcular la inversa de una matriz 3 X 3. El listado 7.5 es el método del adjunto para el cálculo de M, la inversa de la matriz N.

Listado 7.5

```
 500 REM inversa de matriz 3x3
 501 REM Datos de entrada N(3,3)
 502 REM Datos de salida SINGULA
R, M(3,3)
 510 LET SINGULAR=1
 520 LET DET=0: LET NI=2: LET NN
I=3
 530 FOR I=1 TO 3
 540 LET DET=DET+N(1,I)*(N(2,NI)
*N(3,NNI)-N(2,NNI)*N(3,NI))
 550 LET NI=NNI: LET NNI=NI+1: I
F NNI=4 THEN  LET NNI=1
 560 NEXT I
 569 REM El determinante de una
matriz singular es cero, no hay
inversa
 570 IF ABS DET<0.000001 THEN  R
ETURN
 579 REM Calcula M, inversa de N
, por el metodo de los adjuntos
 580 LET NI=2: LET NNI=3
 590 FOR I=1 TO 3
 600 LET NJ=2: LET NNJ=3
 610 FOR J=1 TO 3
 620 LET M(J,I)=(N(NI,NJ)*N(NNI,
NNJ)-N(NI,NNJ)*N(NNI,NJ))/DET
 630 LET NJ=NNJ: LET NNJ=NJ+1: I
F NNJ=4 THEN  LET NNJ=1
 640 NEXT J
 650 LET NI=NNI: LET NNI=NI+1: I
F NNI=4 THEN  LET NNI=1
 660 NEXT I
 670 LET SINGULAR=0
 680 RETURN
```

En esta subrutina, de nuevo, se representan los vectores como matrices unidimensionales (llamados asimismo vectores en terminología matricial); así, B(3) contiene la solución de las ecuaciones, $\boldsymbol{x}$, mientras que K(3) contiene las constantes del plano. Hemos expresado las normales $\boldsymbol{n}_1, \boldsymbol{n}_2$ y $\boldsymbol{n}_3$ en la forma de una matriz 3×3, N, calculando los valores en B a partir de aquélla. Obviamente, si dos de los planos son paralelos entre sí o bien los tres coinciden en una recta, no habrá solución única (es decir, no habrá punto de intersección): en estos casos la variable DET, *determinante* de la matriz N, es cero y la variable SINGULAR = 1, indicando que estamos en el caso de una matriz singular (véase el listado 7.6).

Listado 7.6

```
 100 REM punto de interseccion d
e tres planos

 110 DIM N(3,3): DIM M(3,3): DIM
 K(3): DIM B(3)
 120 LET inv=500
 129 REM Datos de entrada de tre
s planos
 130 PRINT AT 2,10;"COEF.
  CONST."
 140 FOR I=1 TO 3
 150 PRINT AT 2+2*I,0;"PLANO(";I
;")= (    ,    ,    )"
 160 FOR J=1 TO 3
 170 LET I$="TECLEA  N("+STR$ I+
","+STR$ J+") "
 180 INPUT (I$);N(I,J): PRINT AT
 2+2*I,7+4*J;N(I,J)
 190 NEXT J
 200 LET I$="TECLEA  K("+STR$ I+
") "
 210 INPUT (I$);K(I): PRINT AT 2
+2*I,28;K(I)
 220 NEXT I
 229 REM Si la matriz de los vec
tores normales es singular no ha
y interseccion
 230 GO SUB inv
 240 PRINT AT 12,3;"PUNTO DE INT
ERSECCION"
 250 IF SINGULAR THEN  PRINT AT
11,3;"NO EXISTE": STOP
```

```
 259 REM El punto de interseccio
n es (B(1),B(2),B(3))
 260 FOR I=1 TO 3
 270 LET B(I)=0
 280 FOR J=1 TO 3
 290 LET B(I)=B(I)+M(I,J)*K(J)
 300 NEXT J
 310 IF ABS B(I)<0.000001 THEN
LET B(I)=0
 320 PRINT AT 12+2*I,7;"B(";I;")
= ";B(I)
 330 NEXT I
 340 STOP
```

Ejemplo 7.5

Encuéntrese el punto de intersección de los tres planos $(0, 1, 1) \cdot \boldsymbol{x} = 2, (1, 2, 3) \cdot \boldsymbol{x} = 4$ y $(1, 1, 1) \cdot \boldsymbol{x} = 0$.

En forma matricial tendremos

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 0 \end{pmatrix}$$

La inversa de $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix}$ es $\begin{pmatrix} -1 & 0 & 1 \\ 2 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix}$

y por tanto

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 & 0 & 1 \\ 2 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix} \times \begin{pmatrix} 2 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \\ 2 \end{pmatrix}$$

La solución se puede comprobar con facilidad: $(0, 1, 1) \cdot (-2, 0, 2) = 2, (1, 2, 3) \cdot (-2, 0, 2) = 4$ y $(1, 1, 1) \cdot (-2, 0, 2) = 0$, lo que significa que el punto $(-2, 0, 2)$ es común a los tres planos y por tanto es su punto de intersección.

Recta de intersección entre dos planos

Sean los planos

$$\boldsymbol{p} \cdot x = (p_1, p_2, p_3) \cdot x = k_1 \quad \text{y}$$
$$\boldsymbol{q} \cdot x = (q_1, q_2, q_3) \cdot x = k_2$$

Suponemos que los planos no son paralelos y por tanto $\boldsymbol{p} \neq \lambda \boldsymbol{q}$ para cualquier λ. La línea de intersección entre estos dos planos pertenece obviamente a ambos, y por tanto debe ser perpendicular a las normales de ambos planos ($\boldsymbol{p}$ y $\boldsymbol{q}$). Por consiguiente, la dirección de esta recta será $\boldsymbol{d} \equiv \boldsymbol{p} \times \boldsymbol{q}$, y la recta se puede expresar de la forma $\boldsymbol{b} + \mu \boldsymbol{d}$ donde $\boldsymbol{b}$ es cualquier punto de ella. Para resolver completamente el problema, necesitamos averiguar uno de dichos $\boldsymbol{b}$. Si encontramos un punto que es intersección de estos dos planos, junto con un tercero que no es paralelo a ninguno de ellos ni los corta en la recta común, tenemos resuelta la cuestión. Si escogemos un plano con una normal $\boldsymbol{p} \times \boldsymbol{q}$ cumpliremos estas condiciones (recordando además que este producto vectorial está ya calculado). Necesitamos asignar un valor a k_3, pero como este valor es arbitrario, escogeremos $k_3 = 0$ para hacer que este tercer plano pase por el origen. De esta forma, $\boldsymbol{b}$ viene dado por el vector columna

$$\boldsymbol{b} = \begin{pmatrix} p_1 & p_2 & p_3 \\ q_1 & q_2 & q_3 \\ p_2 \times q_3 - p_3 \times q_2 & p_3 \times q_1 - p_1 \times q_3 & p_1 \times q_2 - p_2 \times q_1 \end{pmatrix}^{-1} \times \begin{pmatrix} k_1 \\ k_2 \\ 0 \end{pmatrix}$$

***Ejemplo** 7.6*

Encontrar la recta de intersección entre los planos $(0, 1, 1) \cdot \boldsymbol{x} = 2$ y $(1, 2, 3) \cdot \boldsymbol{x} = 2$. Como $\boldsymbol{p} = (0, 1, 1)$ y $\boldsymbol{q} = (1, 2, 3)$, $\boldsymbol{p} \times \boldsymbol{q} = (1 \times 3 - 1 \times 2, 1 \times 1 - 0 \times 3, 0 \times 2 - - 1 \times 1) = (1, 1, -1)$. Necesitamos la inversa de

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & -1 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} -5 & 2 & 1 \\ 4 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix}$$

y por consiguiente el punto de intersección de los tres planos será

$$\frac{1}{3} \begin{pmatrix} -5 & 2 & 1 \\ 4 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix} \times \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} -6 \\ 6 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 2 \\ 0 \end{pmatrix}$$

y la recta será $(-2, 2, 0) + \mu(1, 1, -1)$.

Se puede comprobar fácilmente este resultado, ya que cualquier punto de la recta debe pertenecer a los dos planos

$$(0, 1, 1) \cdot ((-2, 2, 0) + \mu(1, 1, -1)) = (0, 1, 1) \cdot (-2, 2, 0) + \mu(0, 1, 1)) \cdot$$
$$\cdot (1, 1, -1) = 2 \qquad \text{para todo } \mu \qquad \text{y}$$
$$(1, 2, 2) \cdot ((-2, 2, 0) + \mu(1, 1, -1)) = (0, 1, 1) \cdot (-2, 2, 0) + \mu(1, 2, 3) \cdot$$
$$\cdot (1, 1, -1) = 2 \qquad \text{para todo } \mu$$

En el listado 7.7 se presenta el programa destinado a resolver este problema; nótese que es muy semejante al programa anterior. Obsérvese también que no se usan explícitamente ***p*** y ***q*** en su forma matricial, sino que se almacenan estos valores en las dos primeras filas de la matriz N. La matriz B contiene el vector de base de la recta de intersección, pero ***b*** no está colocado en ninguna matriz ya que sus valores aparecen directamente en la tercera fila de N.

Listado 7.7

```
 100 REM recta de interseccion d
e dos planos
 110 DIM N(3,3): DIM M(3,3): DIM
 K(3): DIM B(3): DIM A$(2)
 120 LET inv=500: LET A$="PQ"
 129 REM Entrada de datos de los
 dos planos
 130 PRINT AT 2,10;"COEF.
  CONST."
 140 FOR I=1 TO 2
 150 PRINT AT 2+2*I,0;"PLANO(";I
;")= (    ,    ,    )"
 160 FOR J=1 TO 3
 170 LET I$="TECLEA  "+A$(I)+"("
+STR$ J+")  "
 180 INPUT (I$);N(I,J): PRINT AT
 2+2*I,7+4*J;N(I,J)
 190 NEXT J
 200 LET I$="TECLEA  K("+STR$ I+
")  "
 210 INPUT (I$);K(I): PRINT AT 2
+2*I,28;K(I)
 220 NEXT I
 229 REM Genera el tercer plano
```

```
 230 LET N(3,1)=N(1,2)*N(2,3)-N(
1,3)*N(2,2)
 240 LET N(3,2)=N(1,3)*N(2,1)-N(
1,1)*N(2,3)
 250 LET N(3,3)=N(1,1)*N(2,2)-N(
1,2)*N(2,1)
 260 LET K(3)=0
 269 REM Si la matriz de vectore
s normales es singular no hay in
terseccion
 270 GO SUB inv
 280 PRINT AT 10,3;"RECTA DE INT
ERSECCION"
 290 IF SINGULAR THEN  PRINT AT
9,3;"NO EXISTE": STOP
 300 PRINT AT 12,2;"VECTOR BASE
    DIRECCION"
 308 REM Recta de interseccion:
 309 REM punto (B(1),B(2),B(3))
y direccion (N(3,1),N(3,2),N(3,3
))
 310 FOR I=1 TO 3
 320 LET B(I)=0
 330 FOR J=1 TO 3
 340 LET B(I)=B(I)+M(I,J)*K(J)
 350 NEXT J
 360 IF ABS B(I)<0.000001 THEN
LET B(I)=0
 370 PRINT AT 12 + 2*I,0;"B(";I;
"( =";B(I)
 380 PRINT AT 12+2*I,20;"D(";I;"
)= ";N(3,I)
 390 NEXT I
 400 STOP
```

Representación de la superficie en forma de función

En el capítulo 3, en nuestro estudio del espacio bidimensional, observamos que las curvas se pueden representar en forma de funciones. Podemos extender esta idea a tres dimensiones cuando estudiamos superficies. La forma más simple de superficies es un plano infinito con normal $\boldsymbol{n} \equiv (n_1, n_2, n_3)$, la cual ya hemos visto expresada como ecuación de coordenadas

$$\boldsymbol{n} \cdot \boldsymbol{x} - k = n_1 \times x + n_2 \times y + n_3 \times z - k = 0$$

Esta expresión se puede reescribir en forma de función para un punto cualquiera $\boldsymbol{x} \equiv (x, y, z)$ de la curva

$$f(\boldsymbol{x}) \equiv f(x, y, z) \equiv n_1 \times x + n_2 \times y + n_3 \times z - k \equiv \boldsymbol{n} \cdot \boldsymbol{x} - k$$

Es ésta una expresión sencilla en función de las variables x, y, z de $\boldsymbol{x}$ que nos permite dividir los puntos del espacio en tres conjuntos, aquellos cuya $f(\boldsymbol{x}) = 0$ (que llamaremos el conjunto cero), aquellos con $f(\boldsymbol{x})$ menor que cero (el conjunto negativo) y aquellos cuya $f(\boldsymbol{x}) > 0$ (el conjunto positivo). Se dice que un punto $\boldsymbol{x}$ pertenece a la superficie si y sólo si pertenece al conjunto cero. Si la superficie divide el espacio en dos mitades, dichas mitades pueden identificarse con los conjuntos positivo y negativo (cada mitad se dice que está *conectada;* es decir, dos puntos cualesquiera pertenecientes a la misma mitad pueden unirse por medio de una curva que no atraviesa la superficie). Hay que tener cierta precaución de nuevo en este sistema: existen muchas curvas que dividen el espacio en más de dos áreas conectadas, resultando entonces imposible establecer una relación de funciones en los conjuntos conectados; por ejemplo, $f(x, y, z) = \cos(y) - \text{sen}(x^2 + z^2)$. Sin embargo, muchas curvas útiles "de buen comportamiento" cumplen esta propiedad, por ejemplo la esfera de radio r

$$f(x) = r^2 - |x|^2$$

es decir,

$$f(x, y, z) = r^2 - x^2 - y^2 - z^2$$

Cuando $f(\boldsymbol{x}) = 0$, $\boldsymbol{x}$ pertenece a la superficie de la esfera, si $f(\boldsymbol{x}) < 0$, $\boldsymbol{x}$ está situado en la parte exterior de la esfera y cuando $f(\boldsymbol{x}) > 0$, $\boldsymbol{x}$ se localiza dentro de la esfera.

La representación de la superficie en forma de función es un concepto muy útil. Se puede utilizar para definir los sistemas de ecuaciones necesarios para calcular las intersecciones de varios objetos. Su uso principal, sin embargo, es determinar si dos puntos cualesquiera, $\boldsymbol{p}$ y $\boldsymbol{q}$, pertenecen a la misma mitad del espacio, dada una superficie que ha dividido éste en dos partes. Para ello, simplemente tenemos que comparar los signos de $f(\boldsymbol{p})$ y $f(\boldsymbol{q})$. Si estos signos son opuestos, cualquier línea que una $\boldsymbol{p}$ y $\boldsymbol{q}$ cortará la superficie. A continuación presentamos un ejemplo.

¿Está un punto en la misma parte del plano que el origen?

Supongamos que el plano está definido (como se hizo anteriormente) por tres puntos no colineales $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$. La ecuación del plano, por tanto, es

$$((\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)) \cdot (\boldsymbol{x} - \boldsymbol{p}_1) = 0$$

Reescribiendo esta expresión en forma de función

$$f(x) \equiv ((\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)) \cdot (\boldsymbol{x} - \boldsymbol{p}_1)$$

Sea un punto cualquiera $\boldsymbol{e}$. Para comprobar si $\boldsymbol{e}$ y el origen están en el mismo semiespacio, necesitamos simplemente comparar $f(\boldsymbol{e})$ con $f(\boldsymbol{O})$, donde $\boldsymbol{O}$ es el origen. Estamos suponiendo que ni $\boldsymbol{O}$ ni $\boldsymbol{e}$ están situados en el plano.

Veremos que esta idea es de gran utilidad en el estudio de algoritmos de líneas ocultas.

Ejemplo *7.7*

Calcular si el origen y el punto (1, 1, 3) están en el mismo semiespacio del plano definido por los puntos (0, 1, 1), (1, 2, 3) y (−2, 3, −1).

Según vimos en el ejemplo 7.4, la representación del plano en forma de función es

$$f(\boldsymbol{x}) \equiv ((-6, -2, 4) \cdot (\boldsymbol{x} - (0, 1, 1))$$

Por tanto

$$f(0, 0, 0) = -(-6, -2, 4) \cdot (0, 1, 1) = -2$$

y

$$f(1, 1, 3) = -(-6, -2, 4) \cdot ((1, 1, 3) - (0, 1, 1)) = 2$$

Así pues, el punto (1, 1, 3) está situado en la parte opuesta del plano con respecto al origen, y si trazamos una recta que una estos dos puntos, cortará el plano en el punto $(1 - \mu)(0, 0, 0) + \mu(1, 1, 3)$ donde $0 < \mu < 1$.

Orden horario o antihorario de los vértices de un polígono convexo orientado en el espacio bidimensional

Suponemos en principio que el polígono es un triángulo definido por los tres vértices $\boldsymbol{p}_1 \equiv (x_1, y_1)$, $\boldsymbol{p}_2 \equiv (x_2, y_2)$ y $\boldsymbol{p}_3 \equiv (x_3, y_3)$. Aun cuando estos puntos están en el espacio bidimensional, nosotros los supondremos en el plano x/y asignando a todos ellos un valor cero en coordenada z. Definiremos como norma el sentido de los lados del polígono como $(\boldsymbol{p}_2 - \boldsymbol{p}_1)$, $(\boldsymbol{p}_3 - \boldsymbol{p}_2)$ y $(\boldsymbol{p}_1 - \boldsymbol{p}_3)$. Al estar todas estas rectas en el plano x/y junto con el origen, sabemos que para todo $i = 1, 2$ ó 3 y para algunos números reales $\boldsymbol{r}_i$ dependiendo de i se cumple que

$$(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i) \times (\boldsymbol{p}_{i+2} - \boldsymbol{p}_{i+1}) = (0, 0, \boldsymbol{r}_i)$$

Esto es debido a que el vector producto es perpendicular al plano x/y y por tanto su única coordenada no nula es la coordenada z. La suma de los subíndices se realiza en módulo 3. Si los vértices se toman sistemáticamente, se observará que todos los signos de estos valores $\boldsymbol{r}_i$ son siempre idénticos; además, lo que es más importante, si los $\boldsymbol{p}_i$ están dados en sentido horario, todos los $\boldsymbol{r}_i$ serán negativos,

mientras que si los $\boldsymbol{p}_i$ son antihorarios, los $\boldsymbol{r}_i$ serán positivos. Si tenemos un polígono convexo orientado, para saber si está dado en sentido horario o antihorario necesitaremos considerar únicamente sus tres primeros vértices. Observaremos más adelante que esta técnica es de una importancia trascendental cuando se trabaja con algoritmos de líneas y superficies ocultas. El listado 7.8 es un ejemplo que nos permite decidir si tres vértices ordenados en un espacio bidimensional forman un triángulo en sentido antihorario.

Listado 7.8

```
 100 REM orientacion de un trian
gulo en 2-D
 110 DIM X(3): DIM Y(3)
 120 LET J$ = "INTRODUZCA COORDE
NADAS DEL TRIANGULO"
 130 FOR I = 1 TO 3
 140 LET I$ ="VERTICE(" + STR$ I
 +")=("
 150 INPUT (J$ + I$); X(I); ",";
Y(I);")"
 160 PRINT AT 2 + 2 *I,0;I$;X(I)
;",";Y(I);")"
 170 NEXT I
 180 PRINT AT 12,0;"EL TRIANGULO
 ES:"
 188 REM (DX1, DY1)Es un vector
de direccion en 2-D que une el p
unto 1 al punto 2.
 189 REM (DX2, DY2)Es un vector
de direccion en 2-D que une el p
unto 2 al punto 3.
 190 LET DX1 = X(2) - X(1): LET
DY1 = Y(2) - Y(1)
 200 LET DX2= X(3) - x(2): LET D
Y2 = Y(3) - Y(2)
 209 REM utiliza el producto vec
torial en 3-D para comprobar la
orientacion del triangulo
 210 IF DX1*DY2 - DX2*DY1 > 0  T
HEN PRINT "ANTI-";
 220 PRINT "HORARIO": STOP
```

Ejemplo 7.8

Comprobar si el polígono dado en el ejemplo 3.4 es antihorario. Los vértices (considerados en tres dimensiones) son (1, 0, 0), (5, 2, 0), (4, 4, 0) y (−2, 1, 0). Las direcciones de los lados son (4, 2, 0), (−1, 2, 0), (−6, −3, 0) y (3, −1, 0).

$$
\begin{aligned}
(4, 2, 0) \times (-1, 2, 0) &= (0, 0, 10) \\
(-1, 2, 0) \times (-6, -3, 0) &= (0, 0, 15) \\
(-6, -3, 0) \times (3, -1, 0) &= (0, 0, 15) \\
(3, -1, 0) \times (4, 2, 0) &= (0, 0, 10)
\end{aligned}
$$

Por tanto, al ser todos ellos positivos, la orientación del polígono es antihoraria. Cuídese en conservar un orden consistente al calcular el producto vectorial; en caso contrario obtendrá una respuesta equivocada. Por ejemplo

$(-6, -3, 0) \times (4, 2, 0) = (0, 0, 0)$ — las líneas son paralelas

o $(-1, 2, 0) \times (3, -1, 0) = (0, 0, -5)$ — hemos escogido los lados en una sencuencia incorrecta

---

Programas completos

I. Listado 7.1 (intersección de recta y plano). Datos requeridos: un vector de base (B(1), B(2), B(3)) y un vector de dirección (D(1), D(2), D(3)) para la recta, una normal (N(1), N(2), N(3)) y una constante K para el plano. Intente los valores (1, 2, 3), (1, 1, −1), (1, 0, 1) y 2 respectivamente.

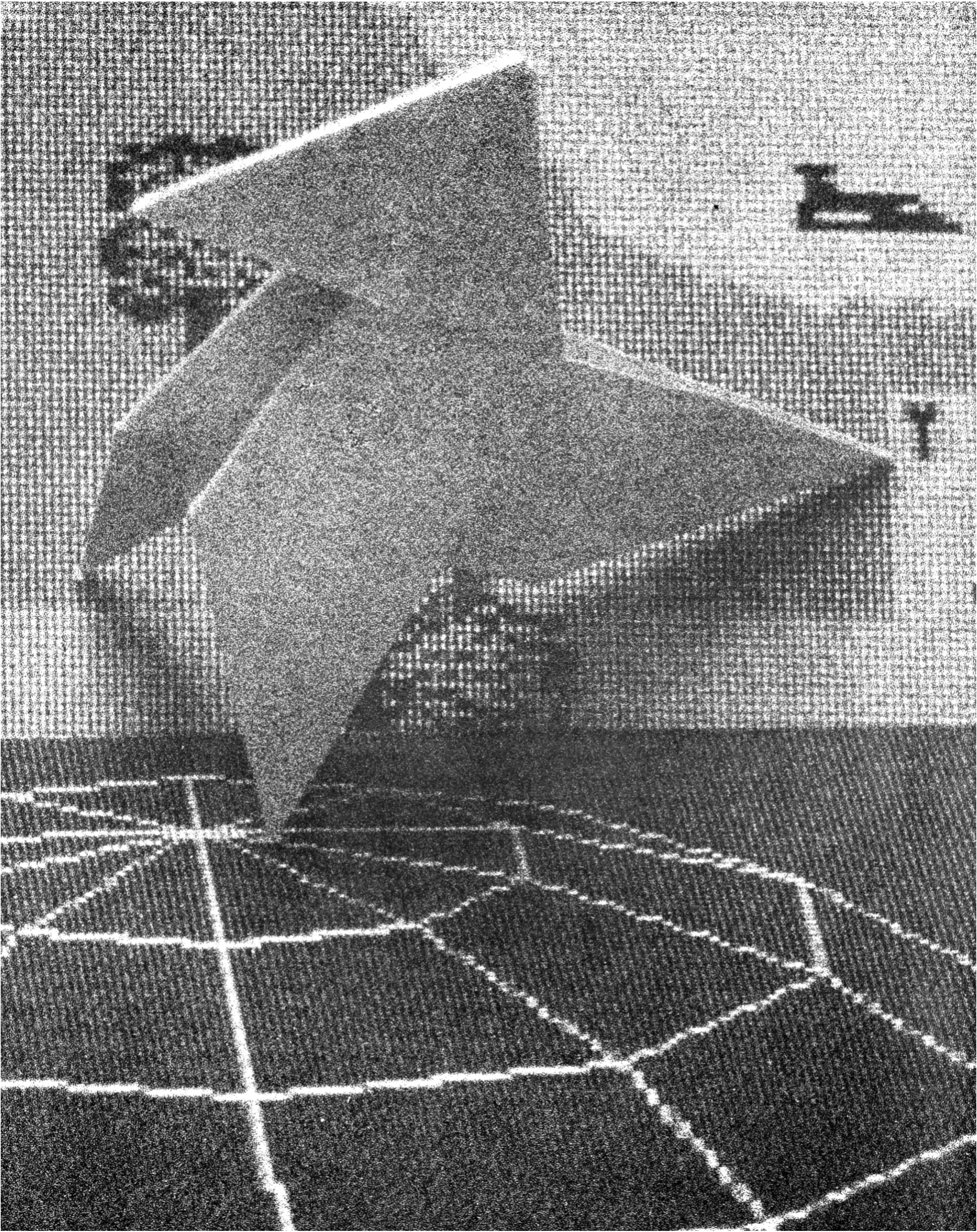
II. Listado 7.2 (intersección de dos rectas). Datos requeridos: vectores de base y dirección para las dos rectas: (B(1), B(2), B(3)) y (D(1), D(2), D(3)), y (C(1), C(2), C(3)) y (E(1), E(2), E(3)) respectivamente. Intente los valores (1, 2, 3), (1, 1, −1) y (−1, 1, 3), (1, 0, 1).

III. Listados 7.3 y 7.4, "programa principal", "producto vectorial y producto escalar" *(vecprod* y *dotprod).* Datos requeridos: dos vectores (L(1), L(2), L(3)) y (M(1), M(2), M(3)). Utilice los valores (1, 2, 3), 1, 1, −1).

IV. Listados 7.5 "inversa de matriz 3 × 3" y 7.6 "intersección de tres planos". Datos requeridos: normal (N(I, 1), N(I, 2), N(I, 3)) y constante K(I) para los tres planos, $1 \leqslant I \leqslant 3$. Sugerimos los valores (1, 2, 3), 0, (1, 1, −1), 1, (1, 0, 1), 2.

V. Listados 7.5 "inversa de matriz 3 × 3" y 7.7 "intersección de dos planos". Datos requeridos: normal (N(I, 1), N(I, 2), N(I, 3)) y constante K(I) para los dos planos, $1 \leqslant I \leqslant 2$. Utilice los valores (1, 2, 3), 0, (1, 1, −1), 1.

VI. Listado 7.8 "orientación de un triángulo en 2-D". Datos requeridos: los vértices (X(I), Y(I)), $1 \leqslant I \leqslant 3$. Intente con los puntos (1, 2), 2, 3) y (−1, 1).



8

Representación matricial de transformaciones en el espacio tridimensional

En el capítulo 4 comprobamos la necesidad de transformar objetos en el espacio bidimensional. Cuando realizamos figuras tridimensionales, habrá también muchas ocasiones en las que sea necesario realizar las transformaciones lineales equivalentes en el espacio de tres dimensiones. Al igual que sucedía en dos dimensiones, hay tres tipos básicos de transformación: traslación, cambio de escala y rotación. Representaremos las transformaciones como matrices cuadradas (que ahora serán 4×4). Un punto cualquiera en el espacio con respecto a los tres ejes de coordenadas, se representa por el vector fila (x, y, z), dicho vector lo consideraremos como un vector columna de cuatro elementos

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Todas las operaciones con matrices (adición, multiplicación por un escalar, trasposición, multiplicación por un vector columna y producto matricial) que estudiamos en el capítulo 4, se pueden extender con facilidad a matrices 4×4 y vectores columna simplemente cambiando el límite superior de los índices a cuatro en lugar de tres. De esta forma, podemos generar una subrutina "mult3" que permita multiplicar dos matrices 4×4. Esta subrutina es equivalente a la subrutina "mult2" (multiplicación2) que usábamos para dos dimensiones. La subrutina multiplica la matriz A por la matriz R dando como resultado la matriz B, la cual se copia a continuación en R. Necesitaremos también la subrutina "identidad3" *("idR3")* que hace la matriz R igual a la matriz identidad (véase el listado 8.1).

Listado 8.1

```
9100 REM multiplicacion3
9101 REM Datos de entrada A(4,4)
,R(4,4)
9102 REM Datos de salida R(4,4)
9110 FOR I=1 TO 3
9120 FOR J=1 TO 4
9130 LET B(I,J)=A(I,1)*R(1,J)+A(
I,2)*R(2,J)+A(I,3)*R(3,J)
9140 NEXT J
9150 LET B(I,4)=B(I,4)+A(I,4)
9160 NEXT I
9170 FOR I=1 TO 3
9180 FOR J=1 TO 4
9190 LET R(I,J)=B(I,J)
9200 NEXT J
9210 NEXT I
9220 RETURN

9300 REM identidad3
9302 REM Datos de salida R(4,4)
9310 DIM R(4,4)
9350 LET R(1,1)=1: LET R(2,2)=1:
 LET R(3,3)=1: LET R(4,4)=1
9370 RETURN
```

Consideremos el caso de una transformación lineal en general sobre puntos del espacio en tres dimensiones. Un punto definido por (x, y, z) "antes" se transforma en (x', y', z') "después" según el siguiente conjunto de *ecuaciones lineales*

$$x' = A_{11} \times x + A_{12} \times y + A_{13} \times z + A_{14}$$

$$y' = A_{21} \times x + A_{22} \times y + A_{23} \times z + A_{24}$$

$$z' = A_{31} \times x + A_{32} \times y + A_{33} \times z + A_{34}$$

a estas tres ecuaciones, añadimos, como siempre, una cuarta ecuación

$$1 = A_{41} \times x + A_{42} \times y + A_{43} \times z + A_{44}$$

la cual, si se ha de cumplir para todos los x, y, z, implica que $A_{41} = A_{42} = A_{43} = 0$ y $A_{44} = 1$.

Las ecuaciones se pueden expresar en forma matricial con un vector columna representando el punto a alcanzar (el punto final) como producto de una matriz por el vector columna que representa al punto inicial

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

De esta forma, si almacenamos la transformación como una matriz, podemos transformar cada punto que se desee considerándolo como un vector columna y multiplicándolo por la izquierda por una matriz de transformación. Tal como sucedía anteriormente, las transformaciones se pueden combinar simplemente realizando la secuencia de transformaciones en el orden adecuado. Si las matrices de transformación son $A, B, C, ..., L, M, N$, la matriz equivalente a la combinación será $N \times M \times L \times \times \cdots \times C \times B \times A$. Tenga siempre presente este orden; como tenemos que multiplicar por la izquierda el vector columna en todas las ocasiones, la primera transformación aparecerá siempre a la derecha de la matriz producto y la última a la izquierda.

Traslación

Se trata de transformar cada punto moviendo un vector que llamaremos TX, TY, TZ. El planteamiento en ecuaciones entre las coordenadas iniciales y las finales es

$$x' = 1 \times x + 0 \times y + 0 \times z + TX$$

$$y' = 0 \times x + 1 \times y + 0 \times z + TY$$

$$z' = 0 \times x + 0 \times y + 1 \times z + TZ$$

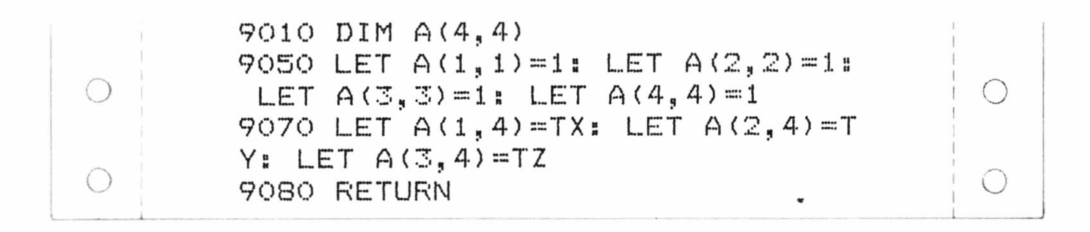
por tanto, la matriz que describe la traslación es

$$\begin{pmatrix} 1 & 0 & 0 & TX \\ 0 & 1 & 0 & TY \\ 0 & 0 & 1 & TZ \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En el listado 8.2 se presenta la subrutina "traslación3", cuya salida es una matriz como la indicada, teniendo como parámetros de entrada TX, TY y TZ.

Listado 8.2

```
9000 REM traslacion3
9001 REM Datos de entrada TX,TY,
TZ
9002 REM Datos de salida A(4,4)
```



```
9010 DIM A(4,4)
9050 LET A(1,1)=1: LET A(2,2)=1:
 LET A(3,3)=1: LET A(4,4)=1
9070 LET A(1,4)=TX: LET A(2,4)=T
Y: LET A(3,4)=TZ
9080 RETURN
```

Cambio de escala

Se trata de transformar la coordenada x de cada punto en una nueva coordenada multiplicada por el factor SX, la coordenada y por el factor SY y la coordenada z por el factor SZ, por tanto

$$x' = \mathrm{SX} \times x + 0 \times y + 0 \times z + 0$$

$$y' = 0 \times x + \mathrm{SY} \times y + 0 \times z + 0$$

$$z' = 0 \times x + 0 \times y + \mathrm{SZ} \times z + 0$$

lo que nos da la matriz

$$\begin{pmatrix} \mathrm{SX} & 0 & 0 & 0 \\ 0 & \mathrm{SY} & 0 & 0 \\ 0 & 0 & \mathrm{SZ} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Generalmente, los factores de escala son positivos; si alguno de los valores es negativo el resultado de la operación es una reflexión así como (posiblemente) un cambio de escala. Por ejemplo, si hacemos $\mathrm{SX} = -1$ y $\mathrm{SY} = \mathrm{SZ} = 1$, los puntos se reflejan en el plano y/z respecto al origen. La subrutina "escala3" del listado 8.3 produce una matriz de escala dados SX, SY y SZ.

Listado 8.3

```
8900 REM escala3
8901 REM Datos de entrada SX,SY,
SZ
8902 REM Datos de salida A(4,4)
8910 DIM A(4,4)
8960 LET A(1,1)=SX: LET A(2,2)=S
Y: LET A(3,3)=SZ
8970 LET A(4,4)=1
8980 RETURN
```

Rotación alrededor de un eje de coordenadas

Antes de considerar el problema de la rotación alrededor de un eje cualquiera $\boldsymbol{p} + \mu\boldsymbol{q}$, abordaremos el caso simplificado de rotación alrededor de un eje de coordenadas.

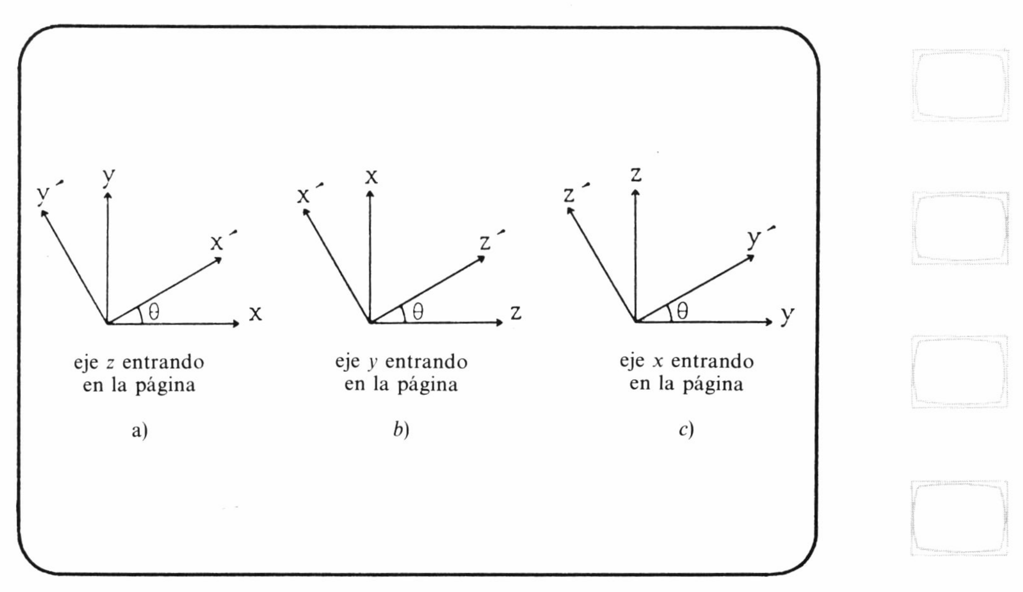


Figura 8.1

a) Rotación de un ángulo θ alrededor de un eje x

Si nos fijamos en la figura 8.1*c*, el eje de rotación es perpendicular a la página (con su parte positiva penetrando en ella, ya que estamos usando la convención de ejes "zurdos"). La figura muestra el punto (x', y', z') resultante de aplicar la transformación a un punto arbitrario (x, y, z). La rotación, como se ve, queda reducida al caso de dos dimensiones en el plano y/z que pasa por el punto; es decir, una vez realizada la rotación, la coordenada x permanece inalterada. Utilizando las ideas explicadas en el capítulo 4 tenemos el siguiente sistema de ecuaciones

$$x' = x$$

$$y' = \cos\theta \times y - \operatorname{sen}\theta \times z$$

$$z' = \operatorname{sen}\theta \times y + \cos\theta \times z$$

y la matriz correspondiente será

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) Rotación de un ángulo θ alrededor del eje y

Observando ahora la figura 8.1*b*, con el eje y positivo penetrando en la página y de acuerdo con nuestra convención, el eje z positivo es horizontal y está a la derecha del origen, mientras que el eje x positivo es vertical. El sistema de ecuaciones será ahora

$$x' = \operatorname{sen}\theta \times z + \cos\theta \times x$$

$$y' = y$$

$$z' = \cos\theta \times z - \operatorname{sen}\theta \times x$$

y la matriz

$$\begin{pmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

c) Rotación de un ángulo θ alrededor del eje z

Observemos ahora la figura 8.1*a*. El sistema de ecuaciones será

$$x' = \cos\theta \times x - \operatorname{sen}\theta \times y$$

$$y' = \operatorname{sen}\theta \times x + \cos\theta \times y$$

$$z' = z$$

lo que equivale a una matriz

$$\begin{pmatrix} \cos\theta & -\operatorname{sen}\theta & 0 & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En el listado 8.4 presentamos el subprograma "rotación3" que produce una matriz como las indicadas, utilizando como parámetros de entrada el ángulo THETA y el número de eje AXIS (AXIS = 1 para el eje x, AXIS = 2 para el eje y y AXIS = 3 para el eje z).

Listado 8.4

```
8600 REM rotacion3
8601 REM Datos de entrada TETA,
AXIS
8602 REM Datos de salida A(4,4)
```

```
8610 DIM A(4,4)
8660 LET A(AXIS,AXIS)=1: LET A(4
,4)=1
8670 LET AX1=AXIS+1: IF AX1=4 TH
EN  LET AX1=1
8680 LET AX2=AX1+1: IF AX2=4 THE
N  LET AX2=1
8690 LET CT=COS THETA: LET ST=SI
N THETA
8700 LET A(AX1,AX1)=CT: LET A(AX
2,AX2)=CT
8710 LET A(AX1,AX2)=-ST: LET A(A
X2,AX1)=ST
8720 RETURN
```

Transformaciones inversas

Consideraremos ahora las transformaciones inversas, antes de abordar el caso general de una transformación de rotación. Se llama transformación inversa a aquella que devuelve los puntos transformados por una transformación dada a su posición original. Si una transformación determinada está representada por una matriz A, la transformación inversa viene dada por la matriz A^{-1}, matriz inversa de A. No es necesario calcular explícitamente la inversa de una matriz utilizando técnicas tales como el método de los adjuntos (listado 7.5): podemos utilizar los listados 8.2, 8.3 y 8.4 con los parámetros deducidos a partir de los parámetros de la transformación original.

1) Una traslación de (TX, TY, TZ) se invierte por medio de otra traslación de (−TX, −TY, TZ).

2) Un cambio de escala SX, SY, SZ se invierte utilizando como factor de escala 1/SX, 1/SY y 1/SZ.

3) Una rotación de un ángulo θ alrededor de un eje dado se invierte con otra rotación de un ángulo $-\theta$ alrededor del mismo eje.

4) Si la matriz de transformación es un producto de un cierto número de matrices de traslación, escala y rotación, $A \times B \times C \quad \cdots \quad L \times M \times N$, la transformación inversa será

$$N^{-1} \times M^{-1} \times L^{-1} \quad \cdots \quad C^{-1} \times B^{-1} \times A^{-1}$$

Rotación de un ángulo γ alrededor de un eje arbitrario $\boldsymbol{p} + \mu\boldsymbol{q}$

Asignemos a $\boldsymbol{p} \equiv (\text{PX, PY, PZ})$ y $\boldsymbol{q} \equiv (\text{QX, QY, QZ})$. Vamos a dividir el problema en una serie de subtareas.

a) Trasladamos todo el espacio de manera que el eje de rotación pase por el origen. Esto se consigue adicionando un vector $-\boldsymbol{p}$ a cada punto del espacio con una matriz F generada por una llamada a "traslación3" con los parámetros TX = = −PX, TY = −PY y TZ = −PZ. Necesitaremos posteriormente la matriz inversa, F^{-1}, que se utilizará con una llamada a "traslación3" con los parámetros PX, PY y PZ. Tras esta transformación el eje de rotación es la línea $\boldsymbol{O} + \mu\boldsymbol{q}$, que pasa por el origen.

$$F = \begin{pmatrix} 1 & 0 & 0 & -\text{PX} \\ 0 & 1 & 0 & -\text{PY} \\ 0 & 0 & 1 & -\text{PZ} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad F^{-1} = \begin{pmatrix} 1 & 0 & 0 & \text{PX} \\ 0 & 1 & 0 & \text{PY} \\ 0 & 0 & 1 & \text{PZ} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) A continuación giramos el espacio alrededor del eje z un ángulo $-\alpha$ donde (ALPHA =) $\alpha = \tan^{-1}$ (QY/QX), que viene dado por la matriz G. Esta matriz se genera llamando a "rotación3", haciendo THETA = −ALPHA y AXIS = 3. La matriz inversa G^{-1} por medio de una "rotación3" con THETA = ALPHA y AXIS = 3. En este momento, el eje de rotación es una recta situada en el plano x/z que pasa por el punto $(v, 0, \text{QZ})$.

$$G = \frac{1}{v}\begin{pmatrix} \text{QX} & \text{QY} & 0 & 0 \\ -\text{QY} & \text{QX} & 0 & 0 \\ 0 & 0 & v & 0 \\ 0 & 0 & 0 & v \end{pmatrix} \quad G^{-1} = \frac{1}{v}\begin{pmatrix} \text{QX} & -\text{QY} & 0 & 0 \\ \text{QY} & \text{QX} & 0 & 0 \\ 0 & 0 & v & 0 \\ 0 & 0 & 0 & v \end{pmatrix}$$

donde v es un número positivo dado por $v^2 = \text{QX}^2 + \text{QY}^2$.

c) Rotamos ahora el espacio alrededor del eje y un ángulo $-\beta$, donde (BETA =) $\beta = \tan^{-1}$ (v/QZ), que viene dado por la matriz H. Esta matriz se genera por una llamada a "rotación3" con los parámetros AXIS = 2 y THETA = −BETA, y la matriz inversa H^{-1}, llamando a "rotación3" con los parámetros AXIS = 2 y THETA = = BETA.

$$H = \frac{1}{w}\begin{pmatrix} \text{QZ} & 0 & -v & 0 \\ 0 & w & 0 & 0 \\ v & 0 & \text{QZ} & 0 \\ 0 & 0 & 0 & w \end{pmatrix} \quad H^{-1} = \frac{1}{w}\begin{pmatrix} \text{QZ} & 0 & v & 0 \\ 0 & w & 0 & 0 \\ -v & 0 & \text{QZ} & 0 \\ 0 & 0 & 0 & w \end{pmatrix}$$

donde w es un número positivo dado por $w^2 = v^2 + \text{QZ}^2 = \text{QX}^2 + \text{QY}^2 + \text{QZ}^2$.

De esta forma, el punto $(v, 0, \text{QZ})$ se ha transformado en el punto $(0, 0, w)$ y por tanto el eje de rotación es el eje z.

d) Rotamos a continuación en el espacio un ángulo γ (GAMMA) alrededor del eje de rotación utilizando la matriz W generada por rotación3 (con AXIS = 3 y THETA = GAMMA).

$$W = \begin{pmatrix} \cos\gamma & -\operatorname{sen}\gamma & 0 & 1 \\ \operatorname{sen}\gamma & \cos\gamma & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e) Finalmente necesitamos devolver el eje de rotación a su posición original, de manera que múltiplicamos por H^{-1}, G^{-1} y, finalmente por F^{-1}.

Así pues, la matriz final P que rota el espacio un ángulo γ alrededor del eje $\boldsymbol{p} + \mu\boldsymbol{q}$ es $P = F^{-1} \times G^{-1} \times H^{-1} \times W \times H \times G \times F$. Naturalmente, algunas de estas matrices se reducirán a la matriz identidad en casos especiales, y pueden por consiguiente despreciarse. Por ejemplo, si el eje de rotación pasa por el origen, F y F^{-1} son idénticas e iguales a la matriz identidad I.

Es posible, por consiguiente, escribir una subrutina especial "rotación general" (listado 8.5) que consigue realizar esta rotación y devuelve la matriz requerida P utilizando como datos de entrada GAMMA, (PX, PY, PZ) y (QX, QY, QZ).

Listado 8.5

```
6800 REM rotacion general
6801 REM Datos de entrada PX,PY,
PZ,QX,QY,QZ,GAMMA,R(4,4)
6802 REM Datos de salida R(4,4)
6809 REM Situa el origen en el e
je de rotacion
6810 LET TX=-PX: LET TY=-PY: LET
 TZ=-PZ: GO SUB tran3: GO SUB mu
lt3
6819 REM Abate el eje de rotacio
n sobre el plano XZ
6820 LET AX=QX: LET AY=QY: GO SU
B angle
6830 LET ALPHA=THETA: LET THETA=
-THETA: LET AXIS=3: GO SUB rot3:
 GO SUB mult3
6839 REM Abate el eje de rotacio
n sobre el eje Z
```

```
6840 LET AX=QZ: LET AY=SQR (QX*Q
X+QY*QY): GO SUB angle
6850 LET BETA=THETA: LET THETA=-
THETA: LET AXIS=2: GO SUB rot3:
GO SUB mult3
6859 REM Rota un angulo GAMMA.al
rededor del eje
6860 LET AXIS=3: LET THETA=GAMMA
: GO SUB rot3: GO SUB mult3
6869 REM Coloca el eje en su pos
icion original
6870 LET AXIS=2: LET THETA=BETA:
 GO SUB rot3: GO SUB mult3
6880 LET AXIS=3: LET THETA=ALPHA
: GO SUB rot3: GO SUB mult3
6890 LET TX=PX: LET TY=PY: LET T
Z=PZ: GO SUB tran3: GO SUB mult3
6900 RETURN
```

Ejemplo 8.1

Explicar qué sucede con los puntos $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ y $(1, 1, 1)$ si se rota el espacio $\pi/4$ radianes alrededor del eje $(1, 0, 1) + \mu(3, 4, 5)$.

Utilizando la teoría anterior tenemos

$$F = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad F^{-1} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$G = \frac{1}{5}\begin{pmatrix} 3 & 4 & 0 & 0 \\ -4 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \quad G^{-1} = \frac{1}{5}\begin{pmatrix} 3 & -4 & 0 & 0 \\ 4 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix} \quad H^{-1} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix}$$

$$W = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix} \text{ y}$$

$$P = \frac{1}{50\sqrt{2}}\begin{pmatrix} 41 + 9\sqrt{2} & -12 - 13\sqrt{2} & -15 + 35\sqrt{2} & -26 + 6\sqrt{2} \\ -12 + 37\sqrt{2} & 34 + 16\sqrt{2} & -20 + 5\sqrt{2} & 32 - 42\sqrt{2} \\ -15 - 5\sqrt{2} & -20 + 35\sqrt{2} & 25 + 25\sqrt{2} & -10 + 30\sqrt{2} \\ 0 & 0 & 0 & 50\sqrt{2} \end{pmatrix}$$

donde $P = F^{-1} \times G^{-1} \times H^{-1} \times W \times H \times G \times F$ es la matriz que representa la transformación requerida. A continuación multiplicamos los vectores columna equivalentes a (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1) y (1, 1, 1) por P, cambiando los vectores columna resultantes a su forma original de fila y sacando un factor común $1/50\sqrt{2}$. El resultado obtenido es $(-26 + 6\sqrt{2}, 32 - 42\sqrt{2}, -10 + 30\sqrt{2})$, $(15 + 15\sqrt{2}, 20 - 5\sqrt{2}, -25 + 25\sqrt{2})$, $(-38 - 7\sqrt{2}, 66 - 26\sqrt{2}, -30 + 65\sqrt{2})$, $(-41 + 41\sqrt{2}, 12 - 37\sqrt{2}, 15 + 55\sqrt{2})$ y $(-12 + 37\sqrt{2}, 34 + 16\sqrt{2}, -20 + 85\sqrt{2})$, respectivamente.

Evidentemente las rotaciones y traslaciones en el espacio deben dejar las posiciones relativas inalterables; en particular, los ángulos formados por los vectores directores deben ser los mismos (esto no es cierto para la transformación de escala, que en general modifica las posiciones relativas). En el sistema original, las tres posiciones relativas de (0, 0, 0) a (1, 0, 0), (0, 1, 0) y (0, 0, 1) respectivamente, son todas ellas perpendiculares entre sí (es decir, el producto escalar de cualquiera de estos pares de direcciones debe ser cero). Por tanto, el producto escalar de las direcciones en el sistema transformado debe ser asimismo cero: los tres vectores de dirección (eliminando el factor $1/50\sqrt{2}$) son $(41 + 9\sqrt{2}, -12 + 37\sqrt{2}, -15 - 5\sqrt{2})$, $(-12 - 13\sqrt{2}, 34 + 16\sqrt{2}, -20 + 35\sqrt{2})$ y $(-15 + 35\sqrt{2}, -20 + 5\sqrt{2}, 25 + 25\sqrt{2})$, y se puede comprobar que el producto escalar de cualquiera de los pares es cero.

De igual forma, el producto escalar del vector de dirección desde el origen al (1, 1, 1) del sistema original, tomado con cualquier otra de las direcciones originales anteriores, da siempre el mismo resultado (= 1). Esto también se cumple en el sistema transformado: la cuarta dirección es $(14 + 31\sqrt{2}, 2 + 58\sqrt{2}, -10 + 55\sqrt{2})$, y calculando el producto escalar de esta dirección con cualquiera de las tres anteriores obtenemos el valor 5000, el cual, teniendo en cuenta el factor común eliminado anteriormente $(1/50\sqrt{2})^2$ da el valor 1.

En el listado 8.6 se presenta un programa que lee el eje de rotación (PX, PY, PZ) + μ(QX, QY, QZ) y el ángulo GAMMA, y rota a cualquier punto (XX, YY, ZZ) alrededor de dicho eje el ángulo establecido.

Listado 8.6

```
 100 REM rotacion de un punto
alrededor de un eje dado
 110 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
 120 DIM P(3): DIM A$(8)
 130 INPUT "PUNTO DEL EJE ","(";
PX;",";PY;",";PZ;")"
 140 PRINT AT 1,0;"PUNTO DEL EJE
",,"(";PX;",";PY;",";PZ;")"
 150 INPUT "VECTOR DIRECTOR DEL
EJE","(";QX;",";QY;",";QZ;")"
 160 PRINT AT 4,0;"VECTOR DIRECT
OR DEL EJE","(";QX;",";QY;",";QZ
;")"
 170 INPUT "ANGULO DE ROTACION "
;GAMMA
 180 PRINT AT 7,0;"ANGULO DE ROT
ACION ";GAMMA
 190 LET mult3=9100: LET idR3=93
00: LET rot3=8600: LET tran3=900
0: LET angle=8800: LET genrot=58
00
 198 REM Calcula R(4,4) para rot
ar el punto un angulo GAMMA alre
dedor
 199 REM del eje de punto (PX,PY
,PZ) y vector director (QX,QY,QZ
)
 200 GO SUB idR3: GO SUB genrot
 210 PRINT AT 14,0;"SE TRANSFORM
A EN"
 219 REM Lee y transforma el pun
to (XX,YY,ZZ)
 220 INPUT "PUNTO","(";XX;",";YY
;",";ZZ;")"
 230 PRINT AT 12,0;"EL PUNTO (";
XX;",";YY;",";ZZ;")",,
 240 LET P(1)=XX*R(1,1)+YY*R(1,2
)+ZZ*R(1,3)+R(1,4)
 250 LET P(2)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
 260 LET P(3)=XX*R(3,1)+YY*R(3,2
)+ZZ*R(3,3)+R(3,4)
```

```
269 REM Salida de datos
270 PRINT AT 16,0;"(";
280 FOR I=1 TO 3: LET A$=STR$ P
(I): FOR J=1 TO 8
290 IF A$(J)<>" " THEN  PRINT A
$(J);
300 NEXT J: IF I<>3 THEN  PRINT
",";
310 NEXT I: PRINT ")",,
320 GO TO 220
```

Ejercicio 8.1

Experimente con las ideas apuntadas. Su resultado puede siempre comprobarse utilizando valores sencillos como hemos hecho arriba. También puede utilizar los listados anteriores para comprobar su respuesta. Es esencial que usted adquiera soltura en el manejo de matrices, y la mejor manera es experimentar con ellas. Al principio cometerá un gran número de errores aritméticos, pero aprenderá pronto a considerar las transformaciones en forma de su representación matricial, lo que resulta de gran ayuda en el estudio del dibujo de objetos en tres dimensiones.

Ejercicio 8.2

Al igual que en el caso bidimensional, habrá observado que la última fila de todas las matrices de transformación es siempre (0, 0, 0, 1), y no tiene utilidad real en los cálculos. Su única función es formar una matriz cuadrada, lo cual es necesario en la definición formal de la multiplicación de matrices. Podemos ajustar esta definición y la de multiplicación de una matriz por un vector columna, de manera que utilicemos únicamente las tres primeras filas de las matrices 4×4 (en el capítulo 4 utilizábamos las dos primeras filas de las matrices 3×3 en los listados 4.2a, 4.3a, 4.4a y 4.5a). Cambie los listados 8.1, 8.2, 8.3 y 8.4 de acuerdo con esta idea.

Ejercicio 8.3

Habrá observado que la subrutina "rotación3" es llamada generalmente con un ángulo THETA generado por "ángulo" que utiliza los valores AX y AY como parámetros de entrada. "Rotación3" calcula el coseno y el seno del ángulo THETA, que equivalen a $AX/\sqrt{(AX^2 + AY^2)}$ y $AY/\sqrt{(AX^2 + AY^2)}$ respectivamente. Escriba otra subrutina de "rotación", *rotxy*, que calcule la matriz de rotación directamente a partir de AX y AY sin necesidad de recurrir a "ángulo".

Obsérvese también que en el comienzo de los listados "rotación3", "escala3" e "identidad3" se procede a poner a cero la matriz reDIMensionándola. Es esta una forma eficiente de hacerlo en el Spectrum. En otros ordenadores habrá de sustituirse

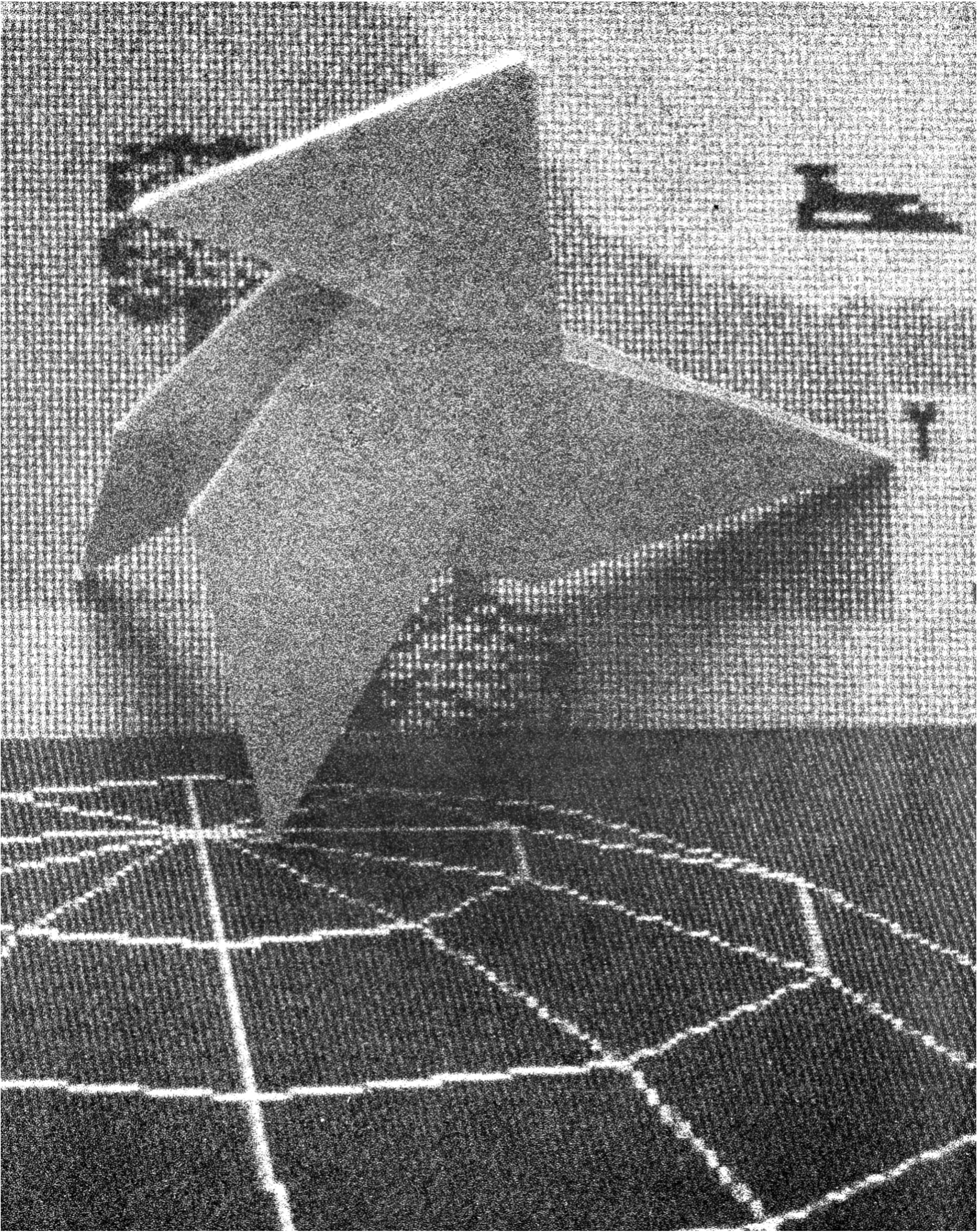
esta parte por una puesta a cero elemento a elemento a través de bucles FOR ... NEXT.

Ejercicio 8.4

Ya se advirtió en el capítulo 4 que algunos autores utilizan vectores fila en lugar de vectores columna multiplicando por la izquierda en lugar de por la derecha. Nosotros utilizamos vectores columna, lo que permite que la matriz de transformación corresponda directamente a los coeficientes de las ecuaciones de transformación. En la otra interpretación se debe utilizar la matriz traspuesta a la obtenida por la coeficientes. Sin embargo es útil conocer la existencia del otro método, por lo que sugerimos que reescriba todos los programas dados en este capítulo (y en el resto del libro). A pesar de todo, recuerde que no es importante el método utilizado, en tanto que usted sea consistente consigo mismo. La notación en vector columna, sin embargo, nos parece más sencilla, y produce menos confusión en las primeras etapas de estudio de esta materia.

Programas completos

I. Todos los listados de este capítulo: 8.1, "multiplicación3" *(mult3)* y "identidad3" *(idR3);* 8.2, "traslación3" *(tran3);* 8.3, "escala3" *(scale3);* 8.4, "rotación3" *(rot3);* 8.5, "rotación general" *(genrot);* 8.6, "programa principal"; y el listado 3.4 "ángulo" *(angle)*. Datos necesarios: vector de base (PX, PY, PZ), vector de dirección (QX, QY, QZ) del eje de rotación y el ángulo GAMMA, junto con un número cualquiera de coordenadas en tres dimensiones (XX, YY, ZZ). Intente con los valores (0, 0, 0), (1, 1, 1) y $\pi/4$, y los puntos (1, 0, 0), (1, 1, 1), (1, 2, 3).



9
Proyecciones ortogonales

Dirigiremos ahora nuestra atención al problema de dibujar objetos tridimensionales en nuestra pantalla gráfica (necesariamente de dos dimensiones). Describimos aquí un método sencillo que es una generalización directa del método introducido en el capítulo 4 para objetos bidimensionales. Volvemos otra vez a utilizar (hasta) tres *posiciones*. Daremos en principio un breve resumen que nos ayude a fijar estas ideas, y a continuación las ampliaremos utilizando un gran número de ejemplos gráficos y numéricos. Comenzamos definiendo un sistema de ejes arbitrarios pero fijos en el espacio que llamaremos el sistema ABSOLUTO. A partir de ahí, como en el espacio de dos dimensiones, consideraremos tres posiciones: 1) INICIAL, 2) ACTUAL y 3) OBSERVADA.

1) Posición INICIAL (SETUP)

La mayor parte de nuestros dibujos se componen de objetos sencillos (por ejemplo, cubos, véase ejemplo 9.1) colocados en una posición y orientación particular en el espacio. Resulta muy ineficiente calcular a mano las coordenadas de cada vértice de estos objetos e introducirlas a continuación en el programa. En lugar de ello, miraremos nuestro objeto buscando una manera sencilla de definirlo respecto a los ejes absolutos, generalmente localizándolo en el origen. La información requerida constará de vértices, coordenadas x, y y z, y quizás líneas que unan pares de vértices o bien caras, áreas poligonales planas limitadas por las líneas mencionadas anteriormente (posteriormente consideraremos el problema de las líneas y superficies ocultas y daremos los algoritmos necesarios). Esta definición elemental del objeto se denomina posición INICIAL. A ella podemos añadir alguna información adicional, como el color del objeto.

2) *Posición ACTUAL*

También denominada posición real. Utilizaremos las técnicas de matrices estudiadas en el último capítulo para generar una matriz que mueva el objeto desde su posición INICIAL a la posición ACTUAL requerida respecto a los ejes absolutos. A esta matriz de paso de posición INICIAL a ACTUAL la denominamos matriz P.

3) *Posición OBSERVADA*

La contemplación de un objeto en el espacio tridimensional requiere un observador, evidentemente (el ojo, y obsérvese que es un solo ojo), colocado en la posición (EX, EY, EZ) relativa a los ejes absolutos, y mirando en una dirección determinada. Esta dirección puede determinarse unívocamente introduciendo otro punto de la línea de visión, que llamaremos (DX, DY, DZ). También podemos inclinar la cabeza, pero ya hablaremos de este problema más adelante. Lo que el ojo ve cuando mira a un objeto tridimensional es una proyección de sus vértices, líneas y caras en un *plano de visión* bidimensional que es normal a la línea de visión. El cálculo de tales proyecciones requiere que preparemos un procedimiento estándar: utilizamos métodos matriciales para transformar todos los puntos del espacio de manera que el ojo se sitúe en el origen, y la línea de visión a lo largo del eje z positivo. Esta posición se denomina posición OBSERVADA, y a la matriz correspondiente que transforma la posición ACTUAL en OBSERVADA la denominamos matriz Q a lo largo del libro. Más adelante veremos con detalle el método de cálculo de Q; por el momento asumiremos que el ojo ya está colocado en el origen mirando al eje z: en este caso, obviamente, la matriz Q es la matriz identidad.

Una vez movidos todos los puntos del espacio a su posición OBSERVADA, observaremos que el plano de visión es paralelo al plano x/y pasando por el origen. Estando ya el ojo colocado en su posición correcta, estamos listos para proyectar el objeto en el plano de visión; obsérvese, sin embargo, que no hemos definido todavía la posición del plano de visión (únicamente conocemos su normal) ni hemos descrito el tipo de proyección del espacio tridimensional sobre el plano. Estas dos condiciones están relacionadas estrechamente. En un capítulo posterior trataremos de la *proyección en perspectiva;* en este capítulo abordaremos la más simple, la proyección *ortogonal.*

Proyección ortogonal

No se puede pedir nada más sencillo. En la proyección ortogonal podemos elegir como plano de visión *cualquier* plano cuya normal esté en la línea de visión. Al realizar la transformación a posición OBSERVADA, el plano de visión será cualquier plano paralelo al plano x/y dado por la ecuación $z = 0$. Por sencillez utilizaremos el plano x/y que pasa por el origen. Los vértices del objeto se proyectan sobre el plano de visión sencillamente haciendo sus coordenadas z igual a cero. Así,

una pareja de puntos cualesquiera en posición observada (x, y, z) y (x, y, z') por ejemplo (donde $z \neq z'$), se proyectan sobre el mismo punto $(x, y, 0)$, en el plano de visión. Seguidamente asociamos los valores x/y del plano con puntos del sistema de coordenadas de la pantalla gráfica (generalmente centrada en la pantalla) utilizando los métodos del capítulo 2. Una vez que se han proyectado los vértices en el plano de visión y de ahí se han pasado a la pantalla, podemos construir la proyección de líneas y caras. Tanto unos como otras están relacionados con los vértices proyectados exactamente de la misma forma que las líneas y caras originales se relacionan con los vértices originales.

Antes de considerar en detalle el caso general, en el que tanto el ojo como la dirección de visión se sitúan arbitrariamente, realizaremos un ejemplo elemental para demostrar la proyección ortogonal.

Ejemplo 9.1

Utilice las ideas anteriores para dibujar la proyección ortogonal de un cubo. A los dibujos como el mostrado en la figura 9.1 los llamaremos *diagramas de líneas* o *esqueletos* (por razones obvias). Consideraremos la posición INICIAL del cubo como ocho vértices situados en $(1, 1, 1), (1, 1, -1), (1, -1, -1), (1, -1, 1), (-1, 1, 1), (-1, 1, -1), (-1, -1, -1)$ y $(-1, -1, 1)$: numeramos estos vértices de 1 a 8. Las doce aristas que forman el cubo unen los vértices 1 y 2, 2 y 3, 3 y 4, 4 y 1, 5 y 6, 6 y 7, 7 y 8, 8 y 1, 1 y 5, 2 y 6, 3 y 7, 4 y 8.

En la figura 9.1a se muestra el ejemplo más sencillo de proyección ortogonal de un cubo, en que el paso de posición INICIAL a ACTUAL es la matriz identidad; es decir, el cubo permanece en su posición INICIAL. Lo que obtenemos es simplemente un cuadrado: los pares de líneas paralelas de la parte delantera y trasera del cubo se proyectan sobre la misma recta en la pantalla. Hemos utilizado un signo "+" en estos diagramas para mostrar la situación del eje z en la posición OBSERVADA (introduciéndose en la pantalla).

La figura 9.1b muestra el mismo cubo dibujado tras haber sufrido las tres transformaciones siguientes en su posición ACTUAL:

a) Rotación del cubo un ángulo $\alpha = -0{,}927295218$ radianes alrededor del eje z (matriz A). Se ha elegido este valor de manera que $\cos\alpha = 3/5$ y $\operatorname{sen}\alpha = -4/5$, lo cual asegura una matriz de rotación exenta de elementos complicados.

b) Traslación por el vector $(-1, 0, 0)$, representado por la matriz B.

c) Rotación de un ángulo $-\alpha$ alrededor del eje y (matriz C).

La matriz de paso de posición INICIAL a ACTUAL será por consiguiente $P = C \times B \times A$, donde

$$A = \begin{pmatrix} 3/5 & 4/5 & 0 & 0 \\ -4/5 & 3/5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 3/5 & 0 & 4/5 & 0 \\ 0 & 1 & 0 & 0 \\ -4/5 & 0 & 3/5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

con lo que P será

$$P = \frac{1}{25}\begin{pmatrix} 9 & 12 & 20 & -15 \\ -20 & 15 & 0 & 0 \\ -12 & -16 & 15 & 20 \\ 0 & 0 & 0 & 25 \end{pmatrix}$$

Así pues, los ocho tríos de coordenadas de posición INICIAL se han transformado en las siguientes ocho coordenadas: $(26/25, -5/25, 7/25)$, $(-14/25, -5/25, -23/25)$, $(-38/25, -35/25, 9/25)$, $(2/25, -35/25, 39/25)$, $(8/25, 35/25, 31/25)$, $(-32/25, 35/25, 1/25)$, $(-56/25, 5/25, 33/25)$, $(-16/25, 5/25, 63/25)$.

Por ejemplo el punto $(1, 1, 1)$ se ha transformado en $(26/25, -5/25, 7/25)$ ya que

$$\frac{1}{25}\begin{pmatrix} 9 & 12 & 20 & -15 \\ -20 & 15 & 0 & 0 \\ -12 & -16 & 15 & 20 \\ 0 & 0 & 0 & 25 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{25}\begin{pmatrix} 26 \\ -5 \\ 7 \\ 25 \end{pmatrix}$$

La matriz Q, matriz de paso de posición ACTUAL a OBSERVADA, es en este caso la matriz identidad; por tanto, las coordenadas proyectadas en el plano de visión serán $(26/26, -5/25)$, $(-14/25\ -5/25)$, $(-38/25, -35/25)$, $(2/25)$, $-35/25)$, $(8/25, 35/25)$. $(-32/25, 35/25)$, $(-56/25, 5/25)$, $(16/25, 5/25)$. Podemos a continuación colocar estos puntos en la pantalla y unirlos con líneas en el mismo orden en que habían sido definidos en el cubo INICIAL.

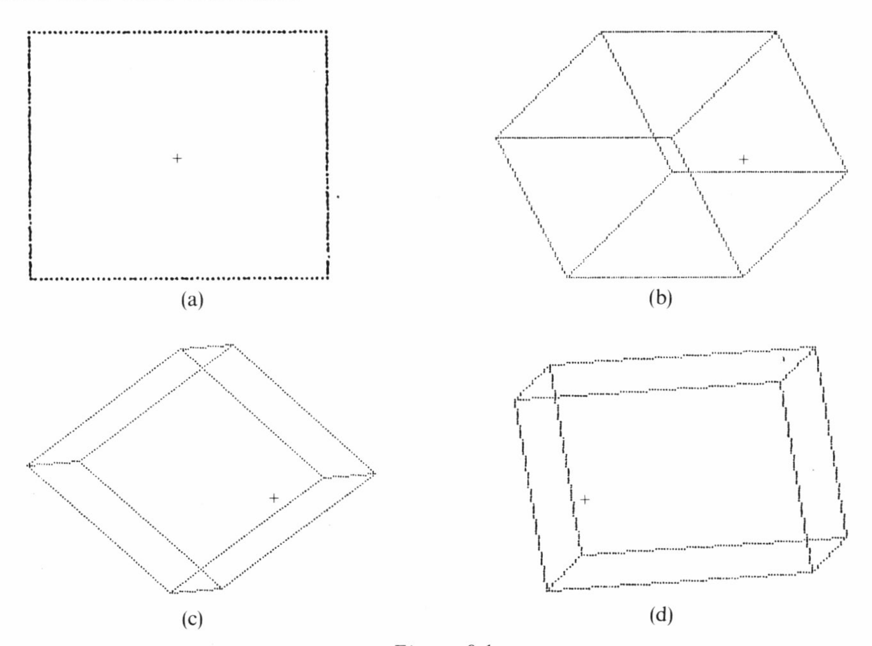


Figura 9.1

Construcción de la matriz Q de paso de posición ACTUAL a OBSERVADA

Suponemos que el ojo está situado en (EX, EY, EZ) con respecto a los ejes absolutos, mirando hacia el punto (DX, DY, DZ). La posición OBSERVADA se consigue con la siguiente secuencia:

1) Una matriz D que traslada todos los puntos del espacio a través de un vector (−DX, −DY, −DZ) de manera que el ojo estará ahora situado en (EX − DX, EY − DY, EZ − DZ) = (FX,FY, FZ), es decir, mirando al origen.

$$D = \begin{pmatrix} 1 & 0 & 0 & -\mathrm{DX} \\ 0 & 1 & 0 & -\mathrm{DY} \\ 0 & 0 & 1 & -\mathrm{DZ} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2) Una matriz E que cambia (FX, FY, FZ) en $(r, 0, \mathrm{FZ})$ rotando el espacio un ángulo $-\alpha$, donde $\alpha = \tan^{-1}$ (FY/FX) alrededor del eje z. $r^2 = \mathrm{FX}^2 + \mathrm{FY}^2$; tomamos el valor positivo de r, $r > 0$.

$$E = \frac{1}{r}\begin{pmatrix} \mathrm{FX} & \mathrm{FY} & 0 & 0 \\ -\mathrm{FY} & \mathrm{FX} & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & r \end{pmatrix}$$

3) A continuación, una matriz F transforma $(r, 0, \mathrm{FZ})$ en (0, 0, −s) rotando el espacio un ángulo $\pi - \theta$ alrededor del eje y, donde $\theta = \tan^{-1}(r/\mathrm{FZ})$. Se puede deducir que $s^2 = r^2 + \mathrm{FZ}^2 = \mathrm{FX}^2 + \mathrm{FY}^2 + \mathrm{FZ}^2$; tomamos el valor positivo de s, s > 0.

$$F = \frac{1}{s}\begin{pmatrix} -\mathrm{FZ} & 0 & r & 0 \\ 0 & s & 0 & 0 \\ -r & 0 & -\mathrm{FZ} & 0 \\ 0 & 0 & 0 & s \end{pmatrix}$$

4) Hasta ahora, la transformación ha colocado el ojo en la posición $(0, 0, -s)$ sobre el eje z negativo mirando hacia el origen. Está situado, además, a una distancia del mismo (s) igual a la distancia que inicialmente había entre el punto (EX, EY, EZ) y el punto (DX, DY, DZ). Nuestra siguiente acción es generar una matriz G que desplaza el ojo al origen.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & s \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5) Como observamos en el ejemplo 9.1, la matriz P se obtenía por multiplicación por la izquierda $P = C \times B \times A$. La generación de la matriz de paso ACTUAL a OBSERVADO, Q, será $Q = G \times F \times E \times D$; por consiguiente, la matriz total de paso de posición INICIAL a OBSERVADA será $R = Q \times P = G \times F \times \times E \times D \times C \times B \times A$. En la figura 9.1c podemos observar el resultado de esta proyección ortogonal. Este dibujo no es totalmente satisfactorio, debido a que la matriz Q coloca el cubo en una orientación arbitraria dentro del plano de visión. Es mejor, pues, realizar una proyección más estándar; una de las formas más comunes de conseguirlo es *mantener la vertical,* es decir, lograr que una línea que era vertical (paralela al eje y) en su posición ACTUAL permanezca vertical tras la transformación con Q en su posición OBSERVADA. Si tomamos la línea vertical desde (DX, DY, DZ) hasta (DX, DY + 1, DZ), observaremos que debido a las peculiaridades de la construcción, la matriz intermedia $K(F \times E \times \times D)$, transforma esta línea en otra que une el punto (0, 0, 0) a (K(1, 2), K(2, 2), K(3, 2)) que llamaremos (p, q, r). Así pues, si añadimos una nueva rotación alrededor del eje z de un ángulo $\beta = \tan^{-1}$ (K(1, 2)/K(2, 2)) $= \tan^{-1}(p/q) = \tan^{-1}$ ($-$FY $\times$ FZ/(s $\times$ FX)) utilizando una matriz H, antes de multiplicar por G, conseguiremos mantener la vertical

$$H = \frac{1}{t}\begin{pmatrix} q & -p & 0 & 0 \\ p & q & 0 & 0 \\ 0 & 0 & t & 0 \\ 0 & 0 & 0 & t \end{pmatrix} \quad \text{donde} \quad t^2 = p^2 + q^2$$

y por tanto

$$H \times \begin{pmatrix} p \\ q \\ r \\ 1 \end{pmatrix} = \frac{1}{t}\begin{pmatrix} q & -p & 0 & 0 \\ p & q & 0 & 0 \\ 0 & 0 & t & 0 \\ 0 & 0 & 0 & t \end{pmatrix} \times \begin{pmatrix} p \\ q \\ r \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ t \\ r \\ 1 \end{pmatrix}$$

En la figura 9.1d podemos ver el resultado final de la transformación, conseguida por la nueva matriz $R = Q \times P = G \times H \times F \times E \times D \times C \times B \times A$; la proyección de la línea que une los puntos (DX, DY, DZ) y (DX, DY + 1, DZ) es ahora la línea que une (0, 0) con (0, t) en la pantalla, es decir, la vertical. La matriz G no afecta los valores x/y; obsérvese que esta técnica funciona en todos los casos excepto cuando (EX, EY, EZ) está situado inicialmente en la vertical sobre (DX, DY, DZ), en cuyo caso, mantener la vertical carece de sentido. El programa "observación3" (listado 9.1) genera la matriz de paso ACTUAL a OBSERVADO, dados (EX, EY, EZ) y (DX, DY, DZ), siguiendo las etapas mostradas aquí; cada etapa multiplica por la izquierda la matriz R: de este modo, al final del proceso R contendrá la matriz original multiplicada por la izquierda por Q. Si deseamos almacenar Q explícitamente, necesitamos hacer en principio R igual a la matriz identidad (utilizando "identidad3"), a continuación llamar a "observación3", y finalmente copiar la matriz R en Q. La sub-

rutina “observación3” se puede reducir radicalmente si suponemos que el ojo siempre mira hacia el origen (DX = DY = DX = 0). Observemos, además, que en proyección ortogonal la posición observada del ojo no tiene por qué estar localizada en el origen; basta con que esté localizada en el eje z: por tanto, podemos reducir aún más la subrutina. Lo que aquí se ha presentado es el caso más general, que resultará de utilidad en posteriores proyecciones en perspectiva.

Listado 9.1

```
8200 REM observacion3
8210 INPUT "(EX,EY,EZ) ";EX;",";
EY;",";EZ
8220 INPUT "(DX,DY,DZ) ";DX;",";
DY;",";DZ
8229 REM Mueve el origen a (DX,D
Y,DZ)
8230 LET TX=-DX: LET TY=-DY: LET
 TZ=-DZ
8240 GO SUB tran3: GO SUB mult3
8249 REM Situa el ojo en el eje
Z negativo, mirando al origen
8250 LET FX=EX-DX: LET FY=EY-DY:
 LET FZ=EZ-DZ
8260 LET AX=FX: LET AY=FY: GO SU
B angle
8270 LET AXIS=3: LET THETA=-THET
A: GO SUB rot3: GO SUB mult3
8280 LET AX=FZ: LET AY=SQR (FX*F
X+FY*FY): GO SUB angle
8290 LET AXIS=2: LET THETA=PI-TH
ETA: GO SUB rot3: GO SUB mult3
8299 REM Lo mantiene en vertical
8300 LET TX=0: LET TY=0: LET TZ=
SQR (FX*FX+FY*FY+FZ*FZ)
8310 LET AX=TZ*FX: LET AY=-FY*FZ
: GO SUB angle
8320 LET AXIS=3: GO SUB rot3: GO
 SUB mult3
8328 REM Situa el ojo en el orig
en:
8329 REM el espacio esta ahora e
n la posicion OBSERVADA
8330 GO SUB tran3: GO SUB mult3
8340 RETURN
```

Podemos ampliar, a nuestra conveniencia, este programa para tratar el caso en que la cabeza está inclinada un ángulo γ respecto a la vertical. Se consigue este último punto añadiendo una nueva rotación $-\gamma$ alrededor del eje z. Así pues, la matriz H deberá girar en este caso alrededor del eje z un ángulo $\beta - \gamma$.

Por último, observemos que la construcción de la matriz de paso ACTUAL a OBSERVADA es independiente, obviamente, de todo lo relacionado con la proyección, con excepción de la posición del ojo, línea de visión e inclinación de la cabeza. De aquí se deduce que si deseamos proyectar una serie de objetos vistos desde la misma posición, bastará con calcular y almacenar la matriz Q una sola vez y utilizarla repetidamente para cada objeto.

Cómo definir un objeto

Abordaremos ahora el problema de representar objetos en el ordenador. No existe en este caso una solución única; en realidad depende de qué se está dibujando y de cómo se desea proyectar. Describiremos en esta sección varias formas de preparar una base de datos para almacenar la información necesaria para dibujar una escena determinada, pero no comentaremos en ningún caso su mayor o menor utilidad. En el resto del libro daremos ejemplos que aporten ideas acerca del valor de un método determinado en una situación particular. Utilizaremos matrices o vectores para almacenar conjuntos grandes de datos; por supuesto, la cantidad de memoria asignada a estas matrices dependerá de la cantidad de información que se necesite para la escena: asegúrese al dimensionar las matrices de que existe memoria suficiente para toda la información; en caso de duda haga una sobreestimación de sus necesidades de almacenamiento.

Vértices. En general, tendremos siempre que definir vértices y otros puntos de referencia en la escena; los almacenaremos como coordenadas x, y y z en los vectores X, Y y Z respectivamente; si el número total no se conoce explícitamente, calcularemos su valor como NOV. Evidentemente, deberá reservarse espacio suficiente en los vectores para no menos de NOV valores. Estos vértices se referirán a la posición INICIAL, ACTUAL u OBSERVADA dependiendo del contexto del problema. También se pueden presentar situaciones (sobre todo en perspectivas) en las que se necesite almacenar las coordenadas x/y de las proyecciones de estos NOV vértices; a este fin dedicaremos los vectores V y W. En el caso de la proyección ortogonal estos dos vectores resultan innecesarios, ya que se pueden utilizar los valores almacenados directamente en los vectores X e Y. La elección de la base de datos, como vemos, depende tanto de las escenas en sí como del tipo de proyección deseada.

Aristas. Podemos almacenar información sobre un cierto número de segmentos que llamaremos NOL; esta información se almacenará en una matriz bidimensional L. Dentro de dicha matriz, la arista I-ésima estará definida por los índices enteros (entre 1 y NOV) de los dos puntos finales del segmento; así pues, en la matriz L almacenaremos los índices correspondientes a la coordenada, en posiciones L(1, I), L(2, I) para el punto I. Evidentemente, los valores reales de las coordenadas de estos puntos se podrán localizar en los vectores X, Y y Z.

Caras. Se denomina cara o faceta a un área poligonal convexa situada en la superficie de un objeto tridimensional; se puede definir de varias formas. La mayor parte de las caras tendrán forma de triángulo o cuadrado, por lo que para reducir al mínimo el gasto de memoria, supondremos que no existen en ningún caso caras con más de seis lados. En la variable NOF almacenaremos el número de caras; estas estarán definidas por los índices de los vértices situados en su perímetro, los cuales guardaremos en la matriz F. El elemento F(I, J) será el índice del vértice I-ésimo de la cara J-ésima. Si la cara no es hexagonal, algunos de estos valores serán *ficticios*, por lo que necesitaremos además el vector H, en el que se anotarán el número de vértices o lados de cada cara. También podemos almacenar C, el código de color de cada cara; utilícese con precaución: recuérdese que el Spectrum permite usar únicamente dos colores en cada bloque como máximo. Otra forma de almacenar las caras es en función de los índices de sus lados, utilizando la propia matriz F, la cual referenciará ahora a la matriz L; en este caso F(I, J) significará el índice de la recta I que pertenece a la cara J. Recuérdese que existen muchos métodos de almacenar representaciones numéricas de las caras y de otros elementos del objeto tridimensional: elija en cada caso la forma que mejor se adapte a su situación particular.

Subrutinas de construcción

Definiremos una subrutina de construcción para cada objeto que se desee dibujar; cada uno de ellos necesitará como parámetros, una matriz *R* que coloque los vértices en su posición, así como cualquier otra información que se necesite acerca del tamaño del objeto (si se va a guardar el objeto en posición inicial, evidentemente no se necesita matriz). La subrutina deberá definir a continuación los vértices, líneas, caras u otros elementos del objeto, y utilizar la matriz *R* para desplazar los vértices del objeto hasta la posición deseada. Dependiendo del contexto del programa, la subrutina podrá bien dibujar el propio objeto, o bien producir una base de datos que contenga esta información. Daremos a continuación ejemplos de ambos métodos.

Si deseamos construir una escena que contenga varios objetos semejantes (en la que los datos estarán en posición ACTUAL u OBSERVADA), no necesitamos fabricar una nueva subrutina de construcción para cada objeto; simplemente deberemos calcular cada vez una nueva matriz *P* de paso de posición INICIAL a ACTUAL, o $Q \times P$ de posición OBSERVADA, dependiendo de la posición a la que deseemos llegar. Lo que sí se necesita, por supuesto, es una nueva subrutina para cada tipo diferente de objetos.

Conseguiremos realizar la escena completa ejecutando un programa principal (listado 9.2), en el que se declaran todas las etiquetas de subrutina, se prepara la pantalla gráfica utilizando valores de entrada HORIZ y VERT, y finalmente se llama a una subrutina "escena3" que distribuye los objetos en el espacio y a continuación los dibuja. El programa principal que se presenta a continuación se utilizará en todos los programas gráficos de tres dimensiones que vienen a continuación; por tanto, piénselo dos veces antes de alterarlo.

Listado 9.2

```
 100 REM programa principal
 110 LET start=9700: LET setorig
in=9600: LET moveto=9500: LET li
neto=9400: LET clip=8400
 120 LET rot3=8600: LET angle=88
00: LET scale3=8900: LET tran3=9
000: LET mult3=9100: LET idR3=93
00
 130 LET scene3=6000: LET look3=
8200
 139 REM Dimensiona y centra el
area de graficos
 140 INPUT "HORIZ ",HORIZ,"VERT
",VERT
 150 GO SUB start
 160 LET XMOVE=HORIZ*0.5: LET YM
OVE=VERT*0.5
 170 GO SUB setorigin
 179 REM Situa la escena
 180 GO SUB scene3
 190 STOP
```

La subrutina "escena3" declara todos los vectores y matrices que se necesitan para almacenar la información de la escena, junto con las matrices A, B, R y (quizá) Q para mover los objetos y colocarlos en su posición. También se declaran todas las etiquetas de subrutina que sean propias de esta escena. Cuando es necesario, se inicializan los valores de NOV y NOL (o NOF) los cuales serán actualizados en las subrutinas de construcción que vengan a continuación. Para dibujar cada objeto en particular, "escena3" debe calcular una matriz P que coloque esta "pieza" en su posición ACTUAL, y a continuación llamar a la subrutina de construcción utilizando la matriz R correcta (ya sea INICIAL a ACTUAL o INICIAL a OBSERVADA). Una vez colocadas todas las "piezas" tenemos terminada la escena. A veces se realiza el dibujo de la proyección dentro de la propia subrutina de construcción, o bien en otras subrutinas diseñadas específicamente para este cometido (por ejemplo, subrutinas con algoritmos de líneas y superficies ocultas): en última instancia, la elección de una u otra forma depende de lo que se está dibujando y de los requerimientos que se exijan a la escena. Encontraremos, como siempre, que el paso de parámetros de matrices a subrutinas está restringido; por ello no generaremos explícitamente P y Q, sino que realizaremos las operaciones necesarias para actualizar R. Cuando se necesite la matriz de paso de ACTUAL a OBSERVADA, la subrutina llamará a "observación3" *(look3)*. Si se desea almacenar Q, deberemos

llamar en principio a identidad *(idR3)*, la cual transforma la matriz *R* en matriz identidad; recuérdese que todas las operaciones matriciales se realizan por medio de las matrices *A* y *R*, usando la matriz *B* para almacenar los resultados intermedios.

Emplee siempre la versión con recorte de "línea" *(lineto)* en sus programas. Es muy fácil escoger valores HORIZ y VERT demasiado pequeños, con lo que parte del objeto a dibujar se situará fuera del área de gráficos; si no hemos previsto la inclusión en el programa de la subrutina de recorte *(clip)*, el programa no funcionará. Existe una razón, sin embargo, para hacer pequeños estos valores: se puede de esta forma ampliar una pequeña parte de la escena, dibujándola a un tamaño muy grande, y recortando todas las líneas exteriores.

En el listado 9.3 presentamos el primer ejemplo de este método, en el cual la subrutina "escena3" construye un dibujo de un cubo como el que se muestra en la figura 9.1d. La escena se puede observar desde cualquier posición manteniendo la vertical. Disponemos también de la subrutina de construcción cubo (*cube*, listado 9.4), que genera los datos necesarios para dibujar un cubo de arista igual a 2. Los vértices, ocho conjuntos de coordenadas, se almacenan en los vectores X, Y y Z. No se requiere en este caso almacenar las aristas del cubo explícitamente; de forma que disponemos la información en una sentencia DATA y dibujamos las líneas directamente. Los datos necesarios para obtener la figura 9.1d son HORIZ = 6, VERT = 4 (EX, EY, EZ) = (−2, 2, 2) y (DX, DY, DZ) = (−1, 0, 0).

Listado 9.3

```
6000 REM escena3/ejemplo 9.1
6010 DIM X(8): DIM Y(8): DIM Z(8
)
6020 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6030 LET cube=6500
6039 REM Calcula la matriz R de
paso a posicion ACTUAL
6040 GO SUB idR3
6050 LET THETA=-0.92729522: LET
AXIS=3: GO SUB rot3: GO SUB mult
3
6060 LET TX=-1: LET TY=0: LET TZ
=0: GO SUB tran3: GO SUB mult3
6070 LET THETA=-THETA: LET AXIS=
2: GO SUB rot3: GO SUB mult3
6079 REM Premultiplica R por la
matriz de paso de ACTUAL a OBSER
VADA
6080 GO SUB look3
6089 REM Llama a la rutina de co
```

```
nstruccion para dibujar el cubo
6090 GO SUB cube
6100 RETURN
```

Listado 9.4

```
6500 REM cubo/lineas (no almacen
ado)
6501 REM IN :R(3,3)
6510 DATA 1,1,1,1,1,-1,1,-1,-1,1
,-1,1,-1,1,1,-1,1,-1,-1,-1,-1,-1
,-1,1
6520 DATA 1,2,2,3,3,4,4,1,5,6,6,
7,7,8,8,5,1,5,2,6,3,7,4,8
6530 RESTORE cube
6539 REM toma los vertices INICI
ALES del cubo y los coloca en PO
SICION OBSERVADA
6540 FOR I = 1 TO 8
6550 READ XX,YY,ZZ
6560 LET X(I) = XX*R(1,1) + YY*R
(1,2) + ZZ*R(1,3) + R(1,4)
6570 LET Y(I) = XX*R(2,1) + YY*R
(2,2) + ZZ*R(2,3) + R(2,4)
6580 LET Z(I) = XX*R(3,1) + YY*R
(3,2) + ZZ*R(3,3) + R(3,4)
6590 NEXT I
6599 REM informacion sobre arist
as: dibuja lineas uniendo pares
de vertices
6600 FOR I = 1 TO 12
6610 READ L1,L2
6620 LET XPT = X(L1): LET YPT =
Y(L1): GO TO moveto
6630 LET XPT = X(L2): LET YPT =
Y(L2): GO TO lineto
6640 NEXT I
6650 RETURN
```

Podemos tener, si lo deseamos, más de un cubo en la escena. Por ejemplo, si reescribimos "escena3" como aparece en el listado 9.5, manteniendo iguales las demás subrutinas, conseguiremos la figura 9.2. Obsérvese que los valores X, Y y Z del primer cubo se borran cuando se introduce la segunda llamada a "cubo". Obsérvese también que, al haber utilizado la misma matriz ACTUAL a OBSERVADA para ambos cubos (que se diferenciaban en sus matrices de paso INICIAL a ACTUAL), es necesario almacenar Q para poderlas utilizar con el segundo cubo. Recuérdese que Q debe multiplicar la matriz P por la izquierda; esta última desplaza el segundo cubo a su posición ACTUAL. Los datos utilizados en la figura 9.2 son HORIZ = 9, VERT = 6 (EX, EY, EZ) = (3, 2, 1) y (DX, DY, DZ) = (0, 0, 0).

Listado 9.5

```
6000 REM escena3/figura 9.2
6010 DIM X(8): DIM Y(8): DIM Z(8
)
6020 DIM A(4,4): DIM B(4,4): DIM
 Q(4,4)
6030 LET cube = 6500
6039 REM dibuja el cubo 1 en pos
icion OBSERVADA
6040 GO SUB idR3: GO SUB look3
6050 GO SUB cube
6059 REM dibuja el cubo 2 en pos
icion OBSERVADA
6060 FOR I = 1 TO 4: FOR J = 1 T
O 4
6070 LET Q(I,J) = R(I,J)
6080 NEXT J: NEXT I
6090 GO SUB idR3
6100 LET TX=3: LET TY=1.5: LET T
Z=2: GO SUB traslacion2: GO SUB
multiplicacion3
6110 FOR I=1  TO 4: FOR J=1 TO 4
6120 LET A(I,J)=Q(I,J)
6130 NEXT J: NEXT I
6140 GO SUB multiplicacion3
6150 GO SUB cube
6160 RETURN
```

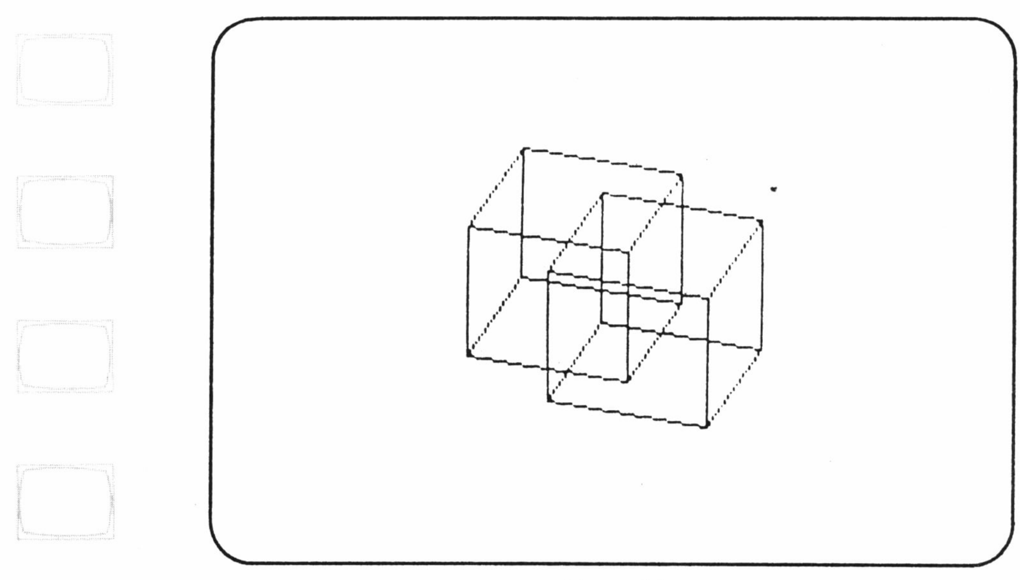


Figura 9.2

Ejercicio 9.1

Generalice la subrutina "cubo" de tal manera que almacene información sobre el tamaño de un bloque rectangular, que permita a la subrutina construir un paralepípedo rectangular de longitud LH, base BH y altura HT: multiplique los valores x del cubo inicial por LH/2, los valores y por HT/2 y los valores z por BH/2.

Debemos hacer notar que la técnica de programación por módulos que explicamos en el libro puede no ser la más eficiente para dibujo de objetos tridimensionales. Sin embargo, es un método descriptivo que permite romper problemas complicados en trozos manejables. Una vez que el lector domine los conceptos, le aconsejamos que *fagocite* nuestros programas y los ofrezca en sacrificio a la diosa Eficiencia. Por el momento,permítanos demostrar la validez de este método modular, dando un nuevo ejemplo que demuestra la rapidez con que se pueden alterar estos programas para producir nuevas escenas y situaciones.

Ejemplo 9.2

Deseamos observar una escena determinada (por ejemplo, la que muestra la figura 9.2) desde distintos ángulos de observación.

En este caso, es mejor almacenar las coordenadas de los vértices de la escena en su posición ACTUAL, en lugar de la posición OBSERVADA; asimismo guardaremos la información sobre las aristas en la matriz L. La subrutina "escena3" (listado 9.6) deberá hacer en primer lugar las variables NOV y NOL igual a cero y colocar los

objetos a continuación en su posición ACTUAL haciendo la matriz *R* igual a *P*. Deberemos, por consiguiente, modificar la subrutina de construcción "cubo" (listado 9.7) con el fin de actualizar la base de datos (obsérvese, sin embargo, que se puede utilizar la misma subrutina para almacenar los vértices en su posición OBSERVADA, ya que sólo necesitamos una matriz diferente $R = Q \times P$). A continuación, para cada ángulo de observación diferente, la subrutina "escena3" deberá borrar la pantalla, hacer *R* igual a la matriz identidad y llamar a "observación3", pasando a continuación el control del programa a una nueva subrutina "dibujo" (listado 9.8), que utiliza la matriz *R* (manteniendo los valores de *Q*, matriz de paso ACTUAL a OBSERVADA) para colocar los puntos en posición OBSERVADA y proyectarlos ortogonalmente en los vectores V y W (no podemos utilizar X e Y porque alteraríamos nuestra base de datos ACTUAL). La subrutina "dibujo" *(drawit)*, etiquetada en "escena3" puede a partir de ese momento utilizar la información del vector L para dibujar la escena en la pantalla.

Listado 9.6

```
6000 REM escena3/figura 9.2 (var
ias vistas)
6010 DIM X(16): DIM Y(16): DIM Z
(16)
6020 DIM V(16): DIM W(16): DIM L
(2,24)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6040 LET cube=6500: LET drawit=7
000
6050 LET NOV=0: LET NOL=0
6059 REM Almacena el primer cubo
 en la posicion ACTUAL
6060 GO SUB idR3
6070 GO SUB cube
6079 REM Almacena el segundo cub
o en la posicion ACTUAL
6080 LET TX=3: LET TY=1.5: LET T
Z=2: GO SUB tran3: GO SUB mult3
6090 GO SUB cube
6099 REM Bucle que genera las di
ferentes vistas
6100 GO SUB idR3: GO SUB look3
6109 REM Dibuja los dos cubos en
 la posicion OBSERVADA
6110 CLS : GO SUB drawit
6120 GO TO 6100
6130 RETURN
```

Listado 9.7

```
6500 REM REM cubo/ vertices y li
neas (almacenadas)
6501 REM Datos de entrada NOV;NO
L,X(NOV),Y(NOV),Z(NOV),L(2,NOL),
R(4,4)
6502 REM Datos de salida NOV,NOL
,X(NOV),Y(NOV),Z(NOV),L(2,NOL)
6510 DATA 1,1,1, 1,1,-1, 1,-1,-1
, 1,-1,1, -1,1,1, -1,1,-1, -1,-1
,-1, -1,-1,1
6520 DATA 1,2, 2,3, 3,4, 4,1, 5,
6, 6,7, 7,8, 8,5, 1,5, 2,6, 3,7,
 4,8
6530 RESTORE cube
6540 LET NV=NOV
6549 REM Lee y almacena los vert
ices en posicion (ACTUAL u OBSER
VADA) usando R
6550 FOR I=1 TO 8
6560 READ XX,YY,ZZ: LET NOV=NOV+
1
6570 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6580 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6590 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6600 NEXT I
6609 REM Lee y almacena la linea
 de informacion
6610 FOR I=1 TO 12
6620 READ L1,L2: LET NOL=NOL+1
6630 LET L(1,NOL)=L1+NV: LET L(2
,NOL)=L2+NV
6640 NEXT I
6650 RETURN
```

Listado 9.8

```
7000 REM dibujo
7001 REM Datos de entrada NOV,NO
L,X(NOV),Y(NOV),Z(NOV),L(2,NOL),
P(4,4)
7002 REM Mueve los vertices a la
 posicion observada y dibuja el
objeto
7010 FOR I=1 TO NOV
7020 LET V(I)=X(I)*R(1,1)+Y(I)*R
(1,2)+Z(I)*R(1,3)+R(1,4)
7030 LET W(I)=X(I)*R(2,1)+Y(I)*R
(2,2)+Z(I)*R(2,3)+R(2,4)
7040 NEXT I
7050 FOR I=1 TO NOL
7060 LET L1=L(1,I): LET L2=L(2,I
)
7070 LET XPT=V(L1): LET YPT=W(L1
): GO SUB moveto
7080 LET XPT=V(L2): LET YPT=W(L2
): GO SUB lineto
7090 NEXT I
7100 RETURN
```

Si el observador viaja en una línea recta, mirando siempre en una misma dirección, la matriz Q no necesita ser recalculada cada vez, sino que se puede manipular el espacio inicialmente de manera que el observador mire a lo largo del eje z y utilizar la subrutina "fijaorigen" *(setorigin)* para mover al observador. Conforme vaya adquiriendo experiencia en el dibujo de proyecciones tridimensionales podrá escoger su sistema de construcción y visualización con mayor conocimiento de causa. Es bastante infrecuente que se tenga que recurrir al método completo dado en este capítulo; normalmente se pueden encontrar atajos que limiten la cantidad de programación requerida.

Ejercicio 9.2

Escriba subrutinas de construcción para un tetraedro, una pirámide, etc. Por ejemplo:

a) tetraedro: vértices (1, 1, 1), (1, −1, −1), (−1, 1, −1) y (−1, −1, 1); las aristas van de 1 a 2, de 1 a 3, de 1 a 4, de 2 a 3, de 2 a 4 y de 3 a 4.

b) pirámide de base cuadrada, de lado 1 y de altura HT: vértices (0, HT, 0), (1, 0, 1), (1, 0, −1), (−1, 0, −1) y (−1, 0, 1); las aristas van de 1 a 2, de 1 a 3, de 1 a 4, de 1 a 5, de 2 a 3, de 3 a 4, de 4 a 5 y de 5 a 1.

Ejemplo 9.3

Prepare un dibujo de un objeto plano cualquiera situado en el plano x/y; por ejemplo, una línea que forme el borde exterior de un carácter alfabético o tira de caracteres. Obsérvelo en distintas orientaciones en el espacio tridimensional. Por ejemplo, coloque el objeto en la cara de un cubo: lo único que necesita es ampliar la subrutina "cubo" para incluir en ella la información adicional de vértices y líneas del nuevo símbolo.

Hasta ahora hemos restringido nuestros dibujos al caso de un sencillo cubo. Se ha hecho así intencionadamente, de manera que la eventual complejidad del dibujo no oscureciese el propio método. Por supuesto, estos programas funcionan exactamente igual con un objeto más complicado, con la única limitación de la propia capacidad de almacenamiento del ordenador (y también de tiempo). Lo único que necesitamos para definir objetos más complejos es aumentar el tamaño de nuestras matrices y vectores, teniendo en cuenta que muchos objetos poseen propiedades geométricas que permiten ahorrar memoria en la información a introducir. Observemos el reactor que aparece en la figura 9.3. Este avión posee un plano de simetría longitudinal, que situaremos en el plano y/z y por tanto cada punto (x, y, z) del reactor se corresponde con otro punto $(-x, y, z)$. La figura 9.3 se dibuja utilizando los lis-

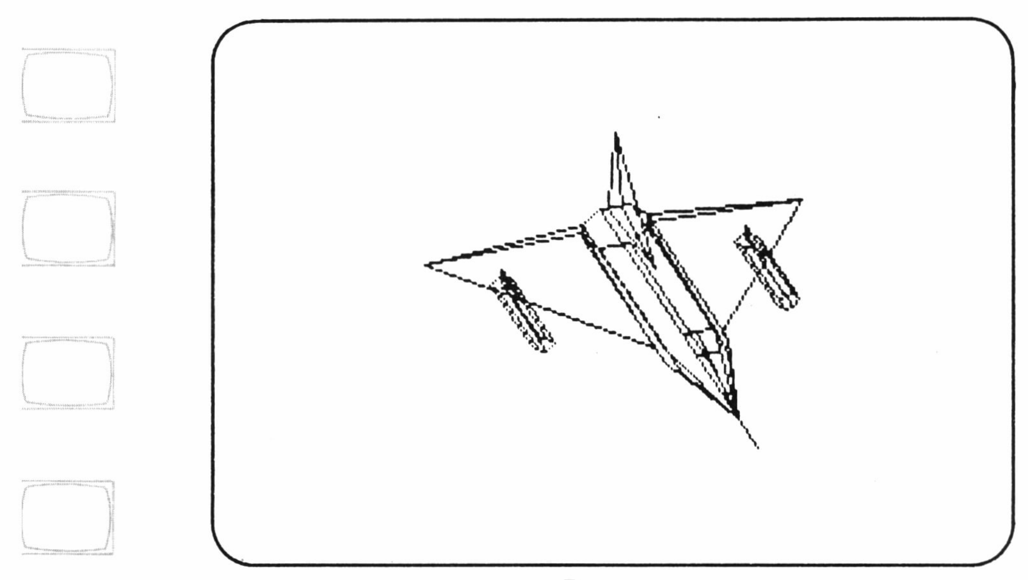


Figura 9.3

tados 9.1, 9.2, 9.3, junto con una subrutina de construcción "reactor" que genera todos los vértices del aeroplano con coordenadas x positivas: así pues, introducimos solamente la mitad de la información requerida para el dibujo completo. Para construir éste, necesitamos también una subrutina "dibujo" (listado 9.9) que dibuja una de las partes del avión, y a continuación cambia de signo todos los valores x, dibujando la parte restante.

La construcción de estas figuras es más sencilla de lo que parece. Simplemente plantee su objeto en distintas partes en una hoja de papel, anote los vértices más importantes y observe los pares de vértices que quedan unidos por líneas. Si realiza el dibujo en un papel cuadriculado, podrá utilizar los valores de coordenadas leídos directamente en la cuadrícula del papel. Los datos necesarios para realizar la figura 9.3 son HORIZ = 160, VERT = 120 (EX, EY, EZ) = (1, 2, 3) y DX, DY, DZ) = = (0, 0, 0).

Listado 9.9

```
6000 REM escena3/reactor
6010 DIM X(37): DIM Y(37): DIM Z
(37)
6020 DIM L(2,46): DIM V(37): DIM
 W(37)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6040 LET jet=6500: LET drawit=70
00
6050 GO SUB idR3: GO SUB look3
6060 GO SUB jet
6070 GO SUB drawit
6080 RETURN

6500 REM reactor
6502 REM Datos de salida NOV,NOL
,X(NOV),Y(NOV),Z(NOV),L(2,NOL)
6510 DATA 0,0,80, 0,0,34, 0,8,32
, 4,8,32
6520 DATA 8,4,32, 8,0,32, 4,-4,3
2
6530 DATA 0,8,-32, 4,8,-32, 8,4,
-32, 8,0,-32
6540 DATA 4,-4,-32, 0,-4,-32, 8,
0,24, 48,0,-32
6550 DATA 8,2,-32, 0,8,0, 2,8,-3
2, 0,32,-32
6560 DATA 28,-4,-24, 30,-2,-24,
```

```
32,-2,-24, 34,-4,-24
6570 DATA 32,-6,-24, 30,-6,-24,
28,-4,8, 30,-2,8
6580 DATA 32,-2,8, 34,-4,8, 32,-
6,8, 30,-6,8
6590 DATA 31,0,-24, 31,-2,-24, 3
1,-2,-12, 31,0,-12
6600 DATA 0,6,40, 3,6,40
6610 DATA 1,2, 2,3, 2,4, 2,5, 2,
6, 2,7, 3,4
6620 DATA 4,9, 5,10, 6,11, 7,12,
 8,9, 9,10, 10,11, 11,12
6630 DATA 12,13, 14,15, 15,10, 1
5,16, 14,16, 17,18, 17,19, 18,19
6640 DATA 20,21, 21,22, 22,23, 2
3,24, 24,25, 25,20, 26,27, 27,28
6650 DATA 28,29, 29,30, 30,31, 3
1,26, 20,26, 21,27, 22,28, 23,29
6660 DATA 24,30, 25,31, 32,33, 3
3,34, 34,35, 35,32, 36,37
6663 REM Fija el vertice y la in
formacion de lineas para la mita
d del reactor
6700 LET NOV=37: LET NOL=46
6710 FOR I=1 TO NOV: READ X(I),Y
(I),Z(I): NEXT I
6720 FOR I=1 TO NOL: READ L(1,I)
,L(2,I): NEXT I
6730 RETURN

7000 REM dibujo/dos mitades del
reactor
7001 REM Datos de entrada NOV,NO
L,X(NOV),Y(NOV),Z(NOV),L(2,NOL),
R(4,4)
7010 LET IS=1
7019 REM Bucle para las dos mita
des del reactor
7020 FOR J=1 TO 2
7030 FOR I=1 TO NOV
7040 LET XX=IS*X(I): LET YY=Y(I)
: LET ZZ=Z(I)
7049 REM Situa los vertices en l
a posicion observada
7050 LET V(I)=XX*R(1,1)+YY*R(1,2
```

```
)+ZZ*R(1,3)+R(1,4)
7060 LET W(I)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
7070 NEXT I
7080 FOR I=1 TO NOL
7090 LET L1=L(1,I): LET L2=L(2,I
)
7100 LET XPT=V(L1): LET YPT=W(L1
): GO SUB moveto
7110 LET XPT=V(L2): LET YPT=W(L2
): GO SUB lineto
7120 NEXT I
7130 LET IS=-1
7140 NEXT J
7150 RETURN
```

Cuerpos de revolución

Hasta el momento, la construcción de objetos en nuestros programas se ha basado en una entrada de datos por medio de sentencias DATA que contuviesen la información necesaria sobre vértices y aristas. Trataremos ahora con un nuevo tipo de objetos, en el cual necesita una pequeña cantidad de información única, siendo el objeto en sí bastante complicado; nos referimos a los cuerpos de revolución, de los que se puede observar un ejemplo en la figura 9.4.

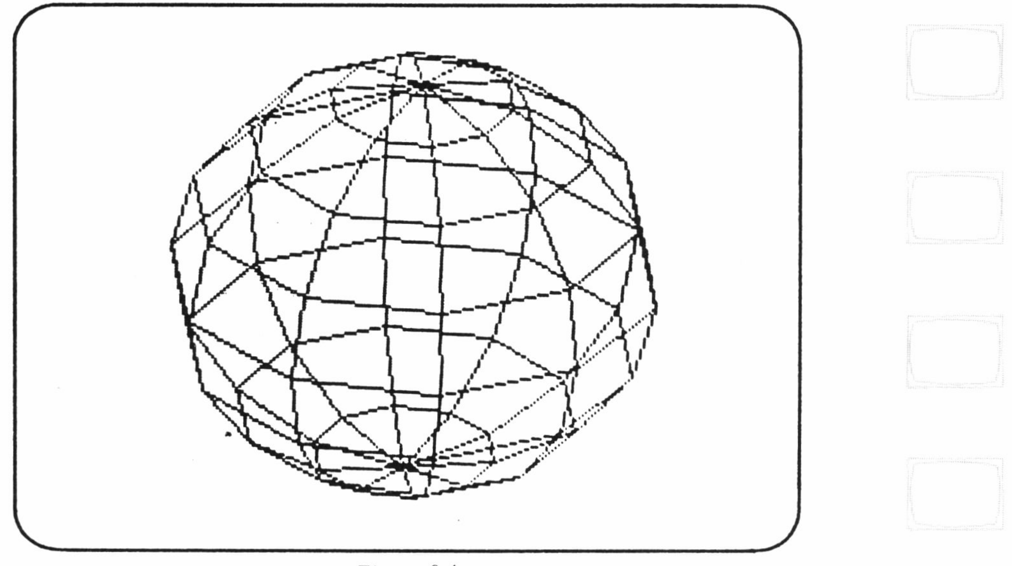


Figura 9.4

El método para dibujar este tipo de objetos es sencillo. Comenzaremos creando una secuencia definida de NUMV líneas en el plano x/y; llamaremos a esta secuencia el conjunto de definición. El paso siguiente es girar dicho conjunto alrededor del eje vertical (eje y) un número determinado de veces, NUMH − 1, con el fin de crear nuevos conjuntos verticales. Las NUMV líneas del conjunto de definición se forman uniendo ordenadamente los NUMV + 1 vértices de la forma (S(I), T(I), 0), donde $1 \leqslant I \leqslant$ NUMV + 1. A partir de este conjunto primario generamos NUMH conjuntos verticales diferentes. Cualquiera de ellos, por ejemplo el J, equivale al conjunto de definición girando un ángulo PHI + 2π(J − I)/NUMH alrededor del eje vertical y, donde PHI (ϕ) es un dato de entrada arbitrario. Una vez definidas las NUMH × × NUMV líneas, todas ellas de planos verticales, introduciremos líneas horizontales además. Las líneas horizontales se consiguen uniendo las diferentes posiciones que cada punto ocupa en los distintos conjuntos. Consideremos el punto (S(I), T(I), 0) que se encuentra al final de un segmento en el conjunto de definición: según vamos rotando alrededor del eje vertical, dicho punto ocupa NUMH posiciones (suponiendo que el punto no se encuentra sobre el eje de rotación)

$$(S(I) \times \cos(\theta + \phi), T(I), S(I) \times \sin(\theta + \phi))$$

donde $\theta = 2\pi(J - I)$ para $1 \leqslant J \leqslant$ NUMH.

Estos NUMH puntos se unen ordenadamente, y el último de ellos (el de posición NUMH-ésima) se une con el primero; de este modo conseguimos el conjunto de líneas horizontales I-ésimo. Así pues, existen (NUMH − n) × NUMV líneas horizontales, donde n es el número de vértices que se encuentran sobre el eje de rotación. El listado 9.10 es una subrutina de construcción que dibuja el "cuerpo de revolución", dados NUMV, NUMH, PHI, el conjunto original de vértices en T y S y la matriz de posicionamiento R. El listado 9.11 es la subrutina correspondiente "escena3" que dibuja el esferoide de la figura 9.4 introduciendo los valores de ocho puntos de un semicírculo en el conjunto de definición: HORIZ = 3.2, VERT = 2.2, PHI = = $\pi/25$, NUMH = 10, NUMV = 8, observado desde (1, 2, 3) mirando hacia (0, 0, 0). En el listado 9.11 se supone que NUMV es menor o igual a quince.

Listado 9.10

```
6000 REM escena3/esferoide
6010 DIM X(32): DIM Y(32)
6020 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6030 DIM S(16): DIM T(16)
6040 LET revbod=6500
6050 INPUT "NUMERO DE LINEAS HOR
IZONTALES",NUMH
6060 INPUT "NUMERO DE LINEAS VER
TICALES",NUMV
```

```
6070 INPUT "ANGULO FI ";PHI
6080 LET THETA=PI/2: LET TD=PI/N
UMV
6089 REM Genera los valores de d
efinicion
6090 FOR I=1 TO NUMV+1
6100 LET S(I)=COS THETA: LET T(I
)=SIN THETA
6110 LET THETA=THETA+TD
6120 NEXT I
6130 GO SUB idR3: GO SUB look3
6140 GO SUB revbod
6150 RETURN
```

Listado 9.11

```
6500 REM cuerpo de revolucion
6501 REM Datos de entrada PHI,NU
MH,NUMV,S(NUMV+1),T(NUMV+1),R(4,
4)
6509 REM Genera el primer juego
de lineas verticales
6510 LET THETA=PHI: LET TD=PI*2/
NUMH
6520 LET N1=NUMV+1: LET C=COS PH
I: LET S=SIN PHI
6530 FOR I=1 TO N1
6540 LET XX=S(I)*C: LET YY=T(I):
 LET ZZ=S(I)*S
6550 LET X(I)=XX*R(1,1)+YY*R(1,2
)+ZZ*R(1,3)+R(1,4)
6560 LET Y(I)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
6570 NEXT I
6579 REM Bucle para el segundo j
uego de lineas verticales
6580 FOR J=1 TO NUMH
6590 LET THETA=THETA+TD: LET C=C
OS THETA: LET S=SIN THETA
6600 FOR I=1 TO N1
6610 LET XX=S(I)*C: LET YY=T(I):
 LET ZZ=S(I)*S
6620 LET X(I+N1)=XX*R(1,1)+YY*R(
```

```
1,2)+ZZ*R(1,3)+R(1,4)
6630 LET Y(I+N1)=XX*R(2,1)+YY*R(
2,2)+ZZ*R(2,3)+R(2,4)
6640 NEXT I
6649 REM Dibuja el primer juego
de lineas verticales
6650 LET XPT=X(1): LET YPT=Y(1):
 GO SUB moveto
6660 FOR I=2 TO N1
6670 LET XPT=X(I): LET YPT=Y(I):
 GO SUB lineto
6680 NEXT I
6689 REM Une los correspondiente
s vertices horizontales en las l
ineas verticales
6690 FOR I=1 TO N1
6700 LET XPT=X(I): LET YPT=Y(I):
 GO SUB moveto
6710 LET XPT=X(I+N1): LET YPT=Y(
I+N1): GO SUB lineto
6719 REM Copia el segundo juego
vertical en el primero y repite
el proceso
6720 LET X(I)=XPT: LET Y(I)=YPT
6730 NEXT I
6740 NEXT J
6750 RETURN
```

Ejemplo 9.4

Siga experimentando con esa técnica; cualquier secuencia de líneas es válida. Pruebe a dibujar un elipsoide: en esencia el problema es el mismo que con el esferoide excepto que en el conjunto de definición se introducen puntos de una semielipse en lugar de puntos de un semicírculo. Los cuerpos producidos no tienen por qué ser convexos. Se pueden cortar unas líneas a otras, cruzar adelante y atrás sobre el eje y y tener valores en x que las mueva arriba y abajo. Esta idea puede ampliarse en el llamado *cuerpo de rotación.* Si tenemos un conjunto de líneas que se mueve alrededor del eje central, los valores y de los puntos no son constantes. Podemos movernos de una forma regular, es decir, con incrementos iguales para cada rotación parcial $2\pi/\mathrm{NUMH}$. Por supuesto, en este caso las líneas podrán realizar más de una rotación completa alreeedor del eje; un ejemplo de este tipo de cuerpos lo tenemos en la figura 9.5. Escriba un programa capaz de realizar una figura semejante a ella.

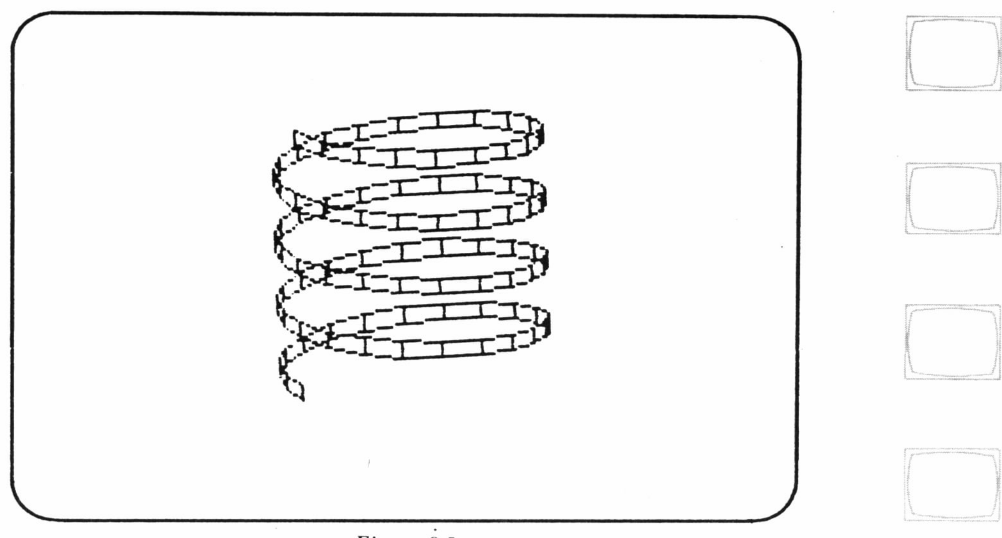


Figura 9.5

Programas completos

En adelante llamaremos a los listados 3.4, "ángulo" *(angle)*; 8.1, "multiplicación3" e "identidad3" *(mult3* e *idR3)*; 8.2, "traslación3" *(tran3)*; 8.3, "escala3" *(scale3)*; 8.4, "rotación3" *(rot3)*; 9.1, "observación3" *(look3)* y 9.2, "programa principal", como un conjunto de programas denominado "rut3".

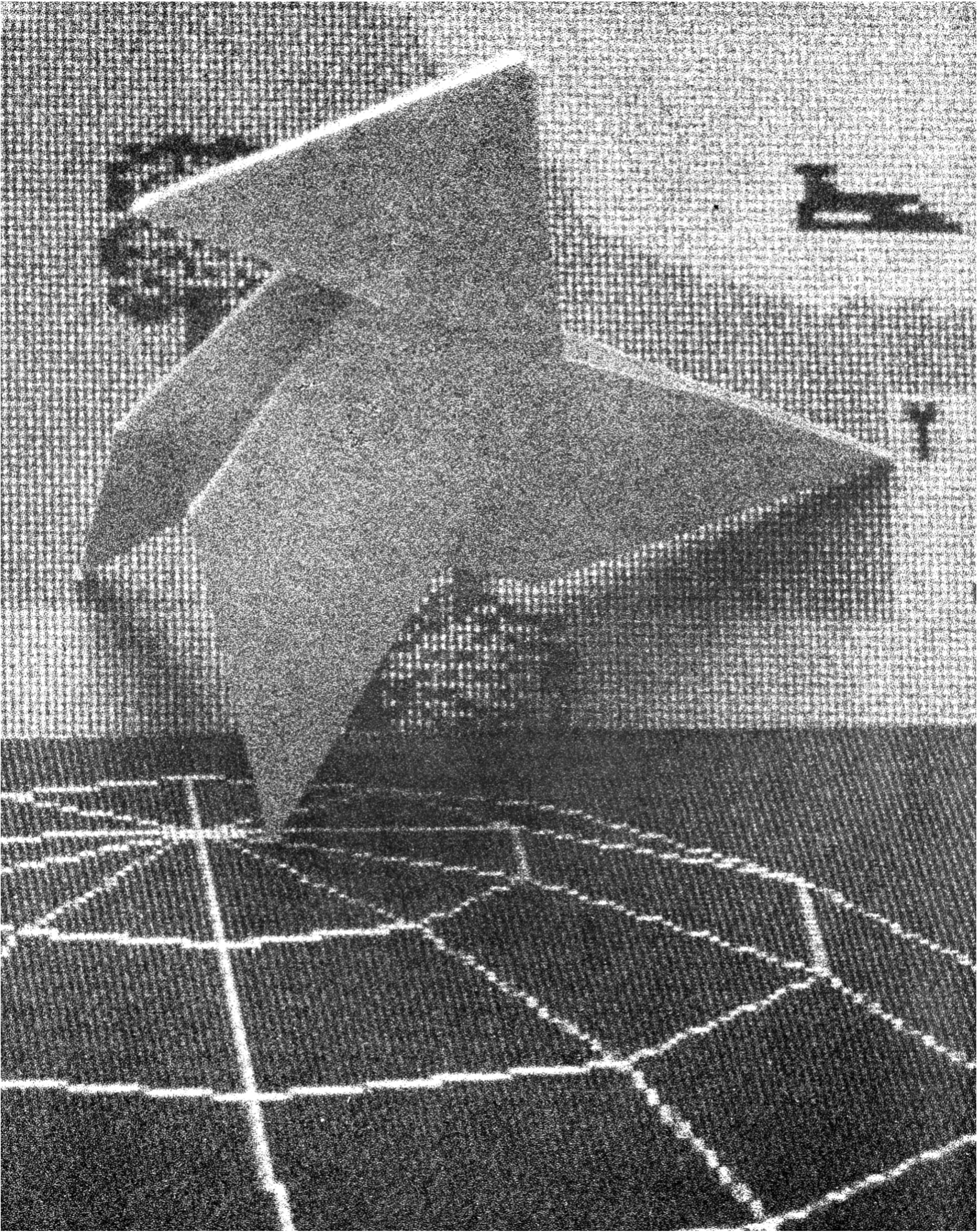
I. "rut1", "rut3" y los listados 9.3, "escena3" *(scene3)* y 9.4 "cubo" *(cube)*. Datos necesarios: HORIZ, VERT (EX, EY, EZ) y (DX, DY, DZ). Intente los valores 6, 4 (1, 2, 3), (−1, 0, 1).

II. "rut1", "rut3" y los listados 9.5, "escena3" *(scene3)* y 9.4, "cubo" *(cube)*. Datos necesarios: HORIZ, VERT, (EX, EY, EZ) y (DX, DY, DZ). Utilice los valores 9, 6, (1, 2, 3), (−1, 0, 1). Realice cambios sistemáticos en uno de los valores manteniendo los demás parámetros constantes.

III. "rut1", "rut3", y los listados 9.6, "escena3" *(scene3)*; 9.7, "cubo" *(cube)* y 9.8, "dibujo" *(drawit)*. Datos necesarios: HORIZ, VERT y entradas repetidas de (EX, EY, EZ) y (DX, DY, DZ). Pruebe los valores 9, 6, y a continuación (1, 2, 3), (−1, 0, 1) y (3, 2, 1), (0, 0, 1). Intente aquí también realizar cambios sistemáticos en uno de los parámetros de entrada.

IV. "rut1", "rut3" y el listado 9.9, "escena3", "reactor", y "dibujo" *(scene3, jet* y *drawit)*. Datos necesarios: HORIZ, VERT, y entradas repetidas de (EX, EY, EZ) y (DX, DY, DZ). Pruebe 160, 120, y a continuación (1, 2, 31), (−1, 0, 30) y (3, 2, 20), (0, 0, 21). También aquí puede intentar cambios sistemáticos.

V. "rut1", "rut3" y los listados 9.10, "escena3" *(scene3)* y 9.11, "cuerpo de revolución" *(revbod)*. Datos: HORIZ; VERT, NUMH, NUMV ($\leqslant$ 15), PHI (EX, EY, EZ) y (DX, DY, DZ). Utilice 3.2, 2.2, 10, 10, 1 (1, 2, 3), (0, 0, 0); (3, 2, 1), (0, 0, 0).



10
Algoritmos sencillos de líneas y superficies ocultas

Ya hemos comenzado a dibujar cubos y otras figuras geométricas compuestas de líneas (lo que llamábamos "esqueletos"). A estas alturas, el lector se sentirá probablemente molesto por la falta de "solidez" de las figuras. Los objetos que estamos acostumbrados a ver en la vida cotidiana son objetos sólidos, en los cuales las caras de la parte frontal del objeto impiden la visión de las caras y aristas posteriores. Para poder transformar nuestras figuras en este tipo de objetos, deberemos introducir un algoritmo de superficies ocultas; o bien un algoritmo de líneas ocultas si deseamos dibujar únicamente aquellas líneas del objeto que sean visibles desde nuestro lugar de observación. Existe un enorme número de algoritmos de este tipo: algunos de ellos muy elementales para situaciones especiales restringidas; otros, muy complicados, para propósito general; con estos últimos seremos capaces de dibujar escenas realmente complejas. La implementación de un algoritmo complicado en un microordenador está limitada por el tiempo de ejecución del programa correspondiente. En el caso del Spectrum, tenemos la limitación adicional de no poder utilizar más de dos colores en un mismo bloque, lo cual a su vez restringe la utilización de las líneas a emplear en el objeto. En cualquier caso, limitando el tipo y número de objetos que aparecen en la escena se puede llegar a conseguir pantallas muy aceptables. En el capítulo 12 presentaremos un algoritmo relativamente complejo; en éste, por el contrario, trataremos con dos tipos especiales de escenas, en los que utilizaremos las propiedades implícitas de estas configuraciones particulares para minimizar el trabajo necesario para la toma de decisiones sobre qué líneas y superficies están ocultas. Más tarde, en este mismo capítulo, daremos un método sencillo para dibujar superficies tridimensionales definidas matemáticamente; para empezar, sin embargo, consideraremos un algoritmo de dibujo de un sólido convexo sencillo en el espacio tridimensional.

Nuestro trabajo en el campo de algoritmo de líneas y superficies ocultas empieza definiendo los NOV vértices de los objetos de la escena en su posición OBSERVADA; almacenaremos estos vértices en los vectores X, Y y Z, como siempre. Sin embargo, en este caso utilizaremos la información de caras en lugar de la de líneas, como hacíamos anteriormente. Las NOF caras se almacenan en una matriz F (necesitaremos además el vector H que indique el número de aristas de cada cara), suponiendo que ninguna cara tiene más de seis aristas con el fin de ahorrar memoria. En caso de que alguna cara tuviese realmente más aristas, podremos dividirla en un conjunto de polígonos menores. Para facilitar la tarea del algoritmo de superficies ocultas, impondremos una restricción sobre el orden de vértices en la matriz F. Los vértices se almacenarán en el orden en que aparecen según vamos bordeando la cara, de tal forma que, visto desde el exterior del objeto, se asigna un sentido antihorario. Evidentemente, este mismo orden desde el interior del objeto tendrá sentido horario. Supondremos también que todas las líneas son uniones de dos caras; si deseamos emplear líneas individuales que no formen parte de caras, las introduciremos como un "polígono" trivial de dos lados.

Orientación de un triángulo tridimensional

Una vez que hemos decidido el objeto a dibujar y lo hemos traducido a vértices y caras, se nos presenta el problema de averiguar si estas caras están orientadas realmente en sentido antihorario. Para resolverlo, simplemente escribimos un programa al efecto. Se puede calcular la orientación de un polígono convexo cualquiera conociendo la de tres de sus vértices, tomados en orden; por consiguiente, necesitamos solamente conocer un triángulo ordenado de vértices de la cara en cuestión. Como ya vimos en el capítulo 7, existe un método sencillo de calcular la orientación de un triángulo bidimensional; nuestro problema, pues, se reduce a adaptar la situación tridimensional a otra en dos dimensiones.

Para simplificar, supondremos que todos los objetos se sitúan en posición INICIAL alrededor del origen. Supondremos también que los planos que contienen a las caras del objeto no pasan por el mismo. Giramos el espacio de manera que uno de los vértices del triángulo en cuestión caiga en el eje z negativo (compárese con la subrutina "observación3" del listado 9.1). Como suponemos que el origen está en el interior del objeto y el eje fuera, simplemente necesitamos proyectar el triángulo transformado de nuevo al plano x/y (es decir, ignorando la coordenada z) y tratarlo como un triángulo bidimensional (de hecho, uno de los tres vértices será $(0, 0)$). En el listado 10.1 se presenta nuestra solución al problema.

Ejercicio 10.1

Reescriba la subrutina de las figuras "esqueleto" del último capítulo, suponiendo que los datos están dados como vértices y caras poligonales antihorarios, y no como líneas. Compruebe los datos de las caras con el programa anterior. La información sobre líneas está contenida implíticamente en la información de caras: recuérdese

que estas líneas son las aristas de cada cara, consideradas como pares de vértices. Obsérvese también que la información de cada arista aparece dos veces, una vez para cada una de las caras adyacentes. Por supuesto, no pretendemos malgastar tiempo dibujando cada línea dos veces. Debido a la forma antihoraria de construcción de las figuras, observaremos que si una línea es la arista que une el vértice I al vértice J de una de las caras, en la cara adyacente la misma línea unirá el vértice J al I. Así pues, cuando la información de una determinada figura se almacene en forma de caras, deberemos dibujar las líneas (aristas) de unión del vértice I al vértice J, si y sólo si I < J.

Listado 10.1

```
 100 REM orientacion de un trian
gulo en 3-D
 110 DIM X(3): DIM Y(3): DIM Z(3
)
 120 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
 130 LET rot3=8600: LET angle=88
00: LET mult3=9100: LET idR3=930
0
 140 LET J$="COORDENADAS DEL TRI
ANGULO          "
 149 REM Transforma el triangulo
 para que todos los vertices se
hallen en el plano XY
 150 FOR I=1 TO 3
 160 LET I$="VERTICE("+STR$ I+")
=("
 170 INPUT (J$+I$);X(I);",";Y(I)
;",";Z(I);")"
 180 PRINT AT 2+2*I,0;I$;X(I);",
";Y(I);",";Z(I);")"
 190 NEXT I
 199 REM Halla la matriz R que s
itua (X(1),Y(1),Z(1)) en el eje
Z negativo
 200 GO SUB idR3
 210 LET AX=X(1): LET AY=Y(1): G
O SUB angle
 220 LET AXIS=3: LET THETA=-THET
A: GO SUB rot3: GO SUB mult3
 230 LET AX=Z(1): LET AY=SQR (X(
```

```
1)*X(1)+Y(1)*Y(1)): GO SUB angle
 240 LET AXIS=2: LET THETA=PI-TH
ETA: GO SUB rot3: GO SUB mult3
 250 FOR I=1 TO 3
 260 LET XX=X(I): LET YY=Y(I): L
ET ZZ=Z(I)
 270 LET X(I)=XX*R(1,1)+YY*R(1,2
)+ZZ*R(1,3)+R(1,4)
 280 LET Y(I)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
 290 NEXT I
 299 REM Comprueba en que sentid
o esta girado el triangulo (ahor
a en 2-D)
 300 PRINT AT 11,0;"SI EL OJO Y
EL ORIGEN ESTAN EN"
 310 PRINT AT 13,0;"CARAS OPUEST
AS ENTONCES"
 320 PRINT AT 15,0;"EL TRIANGULO
 ESTA EN SENTIDO"';
 330 LET DX1=X(2)-X(1): LET DY1=
Y(2)-Y(1)
 340 LET DX2=X(3)-X(2): LET DY2=
Y(3)-Y(2)
 350 IF DX1*DY2-DX2*DY1>0 THEN
PRINT "ANTI";
 360 PRINT "HORARIO"
 370 STOP
```

Un algoritmo de superficies ocultas para cuerpos convexos cerrados sencillos

Llamamos *cuerpo convexo* a aquel en que cualquier segmento que una dos puntos del interior del cuerpo, pertenece al mismo en su totalidad; la definición es una extensión directa del caso bidimensional. Además está cerrado, siendo por tanto imposible pasar del interior al exterior del cuerpo sin atravesar su superficie. Cuando proyectamos ortogonalmente los vértices del objeto en el plano de visión, observamos que la proyección de un polígono convexo de *n* lados en el espacio bidimensional sigue siendo un polígono convexo de *n* lados en el plano de visión (o degenera en una línea). Tomando los vértices proyectados de una cara cualquiera en el mismo orden que los vértices originales, encontraremos que o bien el nuevo polígono bidimensional presenta orientación antihoraria, en cuyo caso estamos viéndolo desde la parte externa de la cara, o bien los nuevos vértices son horarios y estamos observándolos desde el interior. Al ser cerrado el objeto, sólo podremos ver aquellas caras

que contemplamos desde su parte exterior: la visión del resto de las caras se verá bloqueada por el propio objeto. Por tanto nuestra misión se reduce a dibujar únicamente aquellos polígonos que conserven su sentido antihorario; este algoritmo, muy simple, puede implementarse en cualquier subrutina de construcción o en la subrutina "dibujo".

Como ejemplo, presentamos en el listado 10.2 una subrutina de construcción "cubo" preparada para eliminar las líneas ocultas en proyección ortogonal. En este caso no hemos almacenado las caras, sino que tomamos directamente la información por medio de sentencias READ a partir de un bloque DATA y dibujamos aquellas caras que sean visibles inmediatamente. Utilizando este programa se ha realizado la figura 10.1, versión de líneas ocultas de la figura 9.1d. Empleamos todas las subrutinas del capítulo anterior que utilizamos para dibujar la figura 9.1d, con excepción de la subrutina de construcción que preparaba los datos como vértices y caras, y a continuación dibujamos el objeto (estamos sustituyendo el listado 9.4 del programa de dibujo de la figura 9.1d). Los datos utilizados para la figura, naturalmente, siguen siendo los mismos de la figura 9.1d.

Listado 10.2

```
6500 REM cubo/eliminacion de lin
eas ocultas(no almacenadas)
6501 REM Datos de entrada R(4,4)
6510 DATA 1,1,1, 1,1,-1, 1,-1,-1
, 1,-1,1, -1,1,1, -1,1,-1, -1,-1
,-1, -1,-1,1
6520 DATA 1,2,3,4, 5,8,7,6, 1,5,
6,2, 2,6,7,3, 3,7,8,4, 4,8,5,1
6530 RESTORE cube
6539 REM Situa los vertices en l
a posicion OBSERVADA
6540 FOR I=1 TO 8
6550 READ XX,YY,ZZ
6560 LET X(I)=XX*R(1,1)+YY*R(1,2
)+ZZ*R(1,3)+R(1,4)
6570 LET Y(I)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
6580 NEXT I
6590 FOR I=1 TO 6
6598 REM Lee la informacion de c
aras y las dibuja si estan
6599 REM orientadas en sentido a
ntihorario
6600 READ F1,F2,F3,F4
```

```
6610 LET DX1=X(F2)-X(F1): LET DY
1=Y(F2)-Y(F1)
6620 LET DX2=X(F3)-X(F2): LET DY
2=Y(F3)-Y(F2)
6630 IF DX1*DY2-DX2*DY1<0 THEN
GO TO 6690
6640 LET XPT=X(F1): LET YPT=Y(F1
): GO SUB moveto
6650 LET XPT=X(F2): LET YPT=Y(F2
): GO SUB lineto
6660 LET XPT=X(F3): LET YPT=Y(F3
): GO SUB lineto
6670 LET XPT=X(F4): LET YPT=Y(F4
): GO SUB lineto
6680 LET XPT=X(F1): LET YPT=Y(F1
): GO SUB lineto
6690 NEXT I
6700 RETURN
```

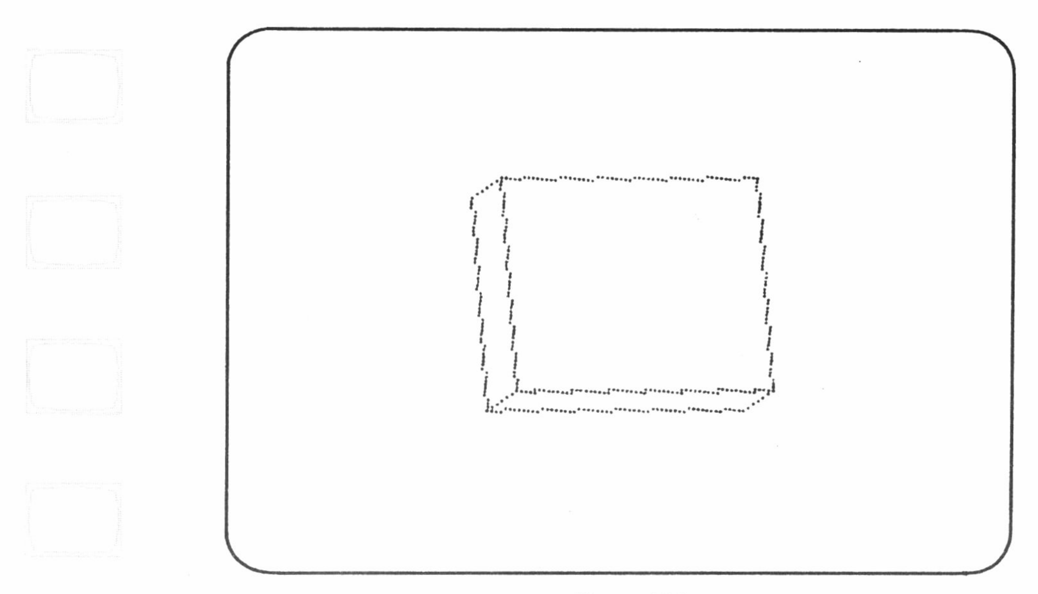


Figura 10.1

Ejercicio ***10.2***

Modifique el listado 10.2, de manera que dibuje un bloque rectangular de longitud LH, base BH y altura HT, donde LH, BH y HT son parámetros de entrada a la subrutina. A continuación realice un programa de eliminación de líneas ocultas

con el mismo. Repita el ejercicio con figuras de tetraedros, pirámides, octaedros, etc. Añada nuevos parámetros al programa para distorsionar las figuras de manera que no sean regulares, aunque sí convexas.

Cuerpos de revolución

Podemos utilizar este método de caras antihorarias y horarios para producir versiones con líneas ocultas de los cuerpos de revolución definidos en el capítulo 9. Partimos de NUMH revoluciones generando NUMV caras en cada movimiento. Nuestras caras serán ahora cuadriláteros o quizá triángulos degenerados; si observamos con cuidado la orientación antihoraria, podemos utilizar el mismo algoritmo. El listado 10.3 es una subrutina que produce la figura 10.2, versión de la figura 9.4 con líneas y superficies ocultas (empleando los mismos datos de entrada). También aquí, debido al diseño modular de nuestros programas, para dibujar la figura 10.2 utilizaremos todas las subrutinas que empleábamos en el capítulo 9, con excepción de "cuerpo de revolución" *(revbod)*. El programa, sin embargo, queda ahora restringido a cuerpos de revolución convexos.

Conforme la subrutina va rotando el conjunto de definición de rectas alrededor del eje vertical, se almacenan los vértices de dos conjuntos consecutivos de rectas verticales. Estos forman aristas verticales de cada *tajada* de caras. Los vértices de estas caras se transforman inmediatamente por medio de la matriz *R* (matriz de paso de posición INICIAL a OBSERVADA) y se almacenan en los vectores X e Y. Con tal configuración de pares de líneas verticales, el primer conjunto de vértices tendrá índices que van desde 1 a NUMV + 1 (= N1), y el segundo desde N1 + 1 hasta 2*N1. La cara I-ésima estará rodeada por cuatro líneas, las dos verticales que unen los vértices I a I + 1, e I + N1 a I + N1 + 1, y los dos horizontales que unen I a I + N1, e I + 1 a I + N1 + 1. Se deben realizar ajustes en caso de que alguno de los vértices originales pertenezca al eje de rotación, en cuyo caso el cuadrilátero degenera formando un triángulo. Escogemos en cada cara el orden de los vértices cuidadosamente de manera que la orientación antihoraria sea la que se observa desde el exterior del objeto. Ello nos permitirá utilizar el sencillo algoritmo enunciado anteriormente para dibujar el objeto eliminando líneas ocultas. Esta técnica es la misma que se usó para dibujar la figura I.1 de la introducción.

Listado 10.3

```
6500 REM cuerpo de revolucion/el
iminacion de lineas ocultas
6501 REM Datos de entrada PHI,NU
MH,NUMV,S(NUMV+1),T(NUMV+1),R(4,
4)
6510 LET THETA=PHI: LET TD=PI*2/
NUMH
```

```
6520 LET N1=NUMV+1: LET C=COS PH
I: LET S=SIN PHI
6529 REM Crea el primer juego de
 lineas verticales
6530 FOR I=1 TO N1
6540 LET XX=S(I)*C: LET YY=T(I):
 LET ZZ=S(I)*S
6550 LET X(I)=XX*R(1,1)+YY*R(1,2
)+ZZ*R(1,3)+R(1,4)
6560 LET Y(I)=XX*R(2,1)+YY*R(2,2
)+ZZ*R(2,3)+R(2,4)
6570 NEXT I
6579 REM  Bucle para el segundo
juego de lineas verticales
6580 FOR J=1 TO NUMH
6590 LET THETA=THETA+TD: LET C=C
OS THETA: LET S=SIN THETA
6600 FOR I=1 TO N1
6610 LET XX=S(I)*C: LET YY=T(I):
 LET ZZ=S(I)*S
6620 LET X(I+N1)=XX*R(1,1)+YY*R(
1,2)+ZZ*R(1,3)+R(1,4)
6630 LET Y(I+N1)=XX*R(2,1)+YY*R(
2,2)+ZZ*R(2,3)+R(2,4)
6640 NEXT I
6648 REM Toma el triangulo en se
ntido antihorario de cada cara e
ntre los 2
6649 REM juegos. Si mantiene la
orientacion al proyectarlo es vi
sible
6650 FOR I=1 TO NUMV
6660 LET F1=I: LET F2=I+1: LET F
3=F2+N1
6670 IF I=NUMV THEN  LET F3=F3-1
6680 LET DX1=X(F2)-X(F1): LET DY
1=Y(F2)-Y(F1)
6690 LET DX2=X(F3)-X(F2): LET DY
2=Y(F3)-Y(F2)
6700 IF DX1*DY2-DX2*DY1<0 THEN
GO TO 6770
6710 LET F3=F2+N1: LET F4=F3-1
6720 LET XPT=X(F1): LET YPT=Y(F1
): GO SUB moveto
6730 LET XPT=X(F2): LET YPT=Y(F2
```

```
): GO SUB lineto
6740 LET XPT=X(F3): LET YPT=Y(F3
): GO SUB lineto
6750 LET XPT=X(F4): LET YPT=Y(F4
): GO SUB lineto
6760 LET XPT=X(F1): LET YPT=Y(F1
): GO SUB lineto
6770 NEXT I
6779 REM Copia el segundo juego
en el primero y repite el proces
o
6780 FOR I=1 TO N1
6790 LET X(I)=X(I+N1): LET Y(I)=
Y(I+N1)
6800 NEXT I
6810 NEXT J
6820 RETURN
```

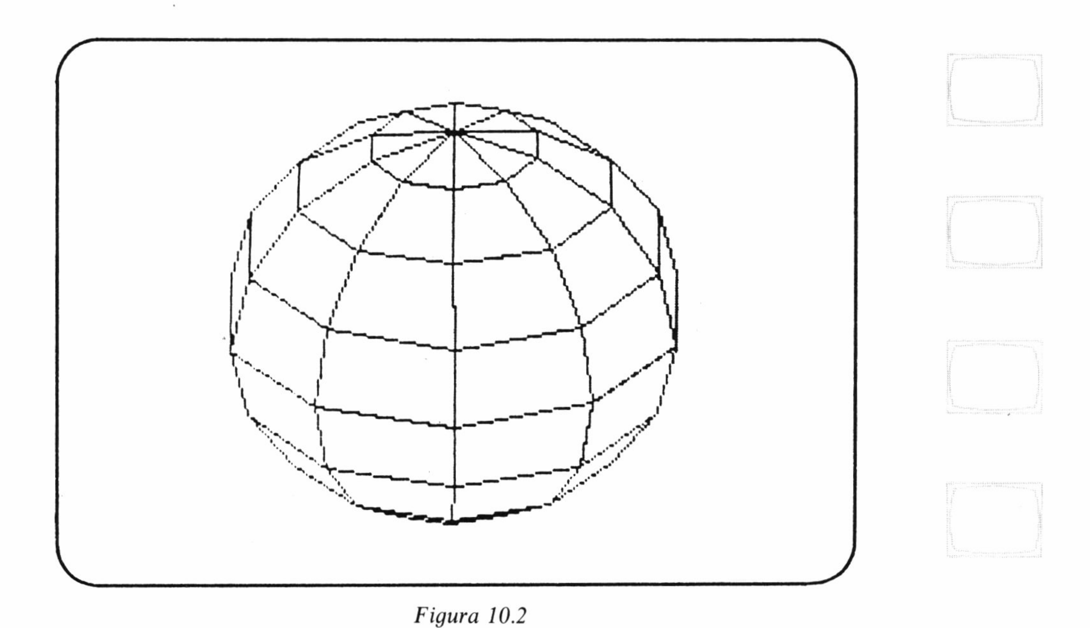


Figura 10.2

Ejercicio 10.3

Experimente con esta técnica. Partiendo de un conjunto cualquiera de líneas inicialmente, se puede originar una figura de este tipo, siempre que el resultado de la rotación alrededor de un eje vertical sea un polígono convexo.

Dibujo de una superficie tridimensional especial

Hasta este momento nos hemos limitado a dibujos de cuerpos convexos; vamos a observar ahora el desarrollo de un tipo de figura no convexa que puede dibujarse sencillamente, debido a la información almacenada sobre su forma particular. Consideremos la construcción de un tipo restringido de superficies tridimensionales en las cuales la coordenada y de cada punto viene dada por una función de x y z, que llamaremos función "f"; la función "f" se almacenará como subrutina del programa; en el listado 10.4 presentamos un ejemplo de este tipo donde la función es $y = 4 \times$ SIN (XZ)/XZ donde XZ = V($x^2 + z^2$). El resultado obtenido se muestra en la figura 10.3. Los datos necesarios son HORIZ = 32, VERT = 22, (EX, EY, EZ) = (3, 2, 1), (DX, DY, DZ) = (0, 0, 0), NX = NZ = 16, XD = ZD = −10 y XT = ZT = 10.

Listado 10.4

```
6000 REM escena3/superficie mat.
6010 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
6020 LET surface=6500: LET drawl
in=7000: LET f=7200
6030 GO SUB idR3: GO SUB look3
6040 GO SUB surface
6050 RETURN
6500 REM superficie
6501 REM Datos de entrada R(4,4)
6510 DIM D(256)
6520 INPUT "NX,XMIN,XMAX ";NX;",
";XMIN;",";XMAX
6530 INPUT "NZ,ZMIN,ZMAX ";NZ;",
";ZMIN;",";ZMAX
6540 LET XDIF=(XMAX-XMIN)/NX
6550 LET ZDIF=(ZMAX-ZMIN)/NZ
6559 REM Dibuja el juego no. 0 d
e lineas de coordenada X fija
6560 LET XX=XMAX: LET ZZ=ZMIN: G
O SUB f
6570 LET XA=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6580 LET YA=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6590 FOR J=1 TO NZ
6600 LET ZZ=ZZ+ZDIF: GO SUB f
6610 LET XB=XX*R(1,1)+YY*R(1,2)+
```

```
ZZ*R(1,3)+R(1,4)
6620 LET YB=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6630 GO SUB drawlin
6640 LET XA=XB: LET YA=YB
6650 NEXT J
6659 REM Dibuja el juego O de li
neas de coordenada Z fija
6660 FOR J=1 TO NX
6670 LET XX=XX-XDIF: GO SUB f
6680 LET XB=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6690 LET YB=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6700 GO SUB drawlin
6710 LET XA=XB: LET YA=YB
6720 NEXT J
6729 REM Varia los valores X en
incrementos NX
6730 LET XS=XMAX
6740 FOR I=1 TO NX
6748 REM Dibuja las partes visib
les de cada linea de coordenada
Z fija:
6749 REM los valores X varian de
 la linea X (I-1) a la (I)
6750 LET ZS=ZMAX
6760 FOR J=1 TO NZ
6770 LET ZS=ZS-ZDIF
6780 LET XX=XS: LET ZZ=ZS: GO SU
B f
6790 LET XA=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6800 LET YA=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6810 LET XX=XS-XDIF: GO SUB f
6820 LET XB=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6830 LET YB=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6840 GO SUB drawlin
6850 NEXT J
6859 REM Dibuja las partes visib
les de las lineas de coordenada
X fija
```

```
6860 LET ZZ=ZMIN: GO SUB f
6870 LET XA=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6880 LET YA=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6890 FOR J=1 TO NZ
6900 LET ZZ=ZZ+ZDIF: GO SUB f
6910 LET XB=XX*R(1,1)+YY*R(1,2)+
ZZ*R(1,3)+R(1,4)
6920 LET YB=XX*R(2,1)+YY*R(2,2)+
ZZ*R(2,3)+R(2,4)
6930 GO SUB drawlin
6940 LET XA=XB: LET YA=YB
6950 NEXT J
6960 LET XS=XS-XDIF
6970 NEXT I
6980 RETURN

7000 REM dibujalineas/ dibuja li
neas entre puntos
7001 REM Datos de entrada XA,YA,
XB,YB,D(256)
7008 REM Partes visibles de la l
inea entre (XA,YA) y (XB,YB)
7009 REM IA e IB son las posicio
nes X del elemento de pantalla d
e los 2 puntos
7010 LET IA=FN X(XA): LET IB=FN
X(XB)
7020 LET Y=YA: LET YD=0
7030 IF IA<>IB THEN  GO TO 7080
7039 REM Si IA=IB dibuja la line
a solo si el segundo punto esta
sobre el primero
7040 PLOT IA,D(IA)
7050 LET IY=FN Y(YB)
7060 IF IY>D(IA) THEN  DRAW 0,IY
-D(IA): LET D(IA)=IY
7070 RETURN
7079 REM Varia la columna del el
emento de pantalla de izquierda
a derecha
7080 LET YD=(YB-YA)/(IB-IA)
7090 FOR K=IA TO IB
7100 LET IY=FN Y(Y)
```

```
7109 REM Si el elemento de panta
lla y es mayor que D, pone a cer
o D
7110 IF D(K)<IY THEN  LET D(K)=I
Y
7120 LET Y=Y+YD
7130 NEXT K
7139 REM Une todos los puntos (I
,D(I)); donde IA<=I<=IB
7140 PLOT IA,D(IA)
7150 FOR K=IA+1 TO IB
7160 DRAW 1,D(K)-D(K-1)
7170 NEXT K
7180 RETURN

7200 REM f/funcion a dibujar
7201 REM Datos de entrada XX,ZZ
7202 REM Datos de salida YY
7210 LET YY=4: LET XZ=SQR (XX*XX
+ZZ*ZZ)
7220 IF XZ>0.000001 THEN  LET YY
=4*SIN (XZ)/XZ
7230 RETURN
```

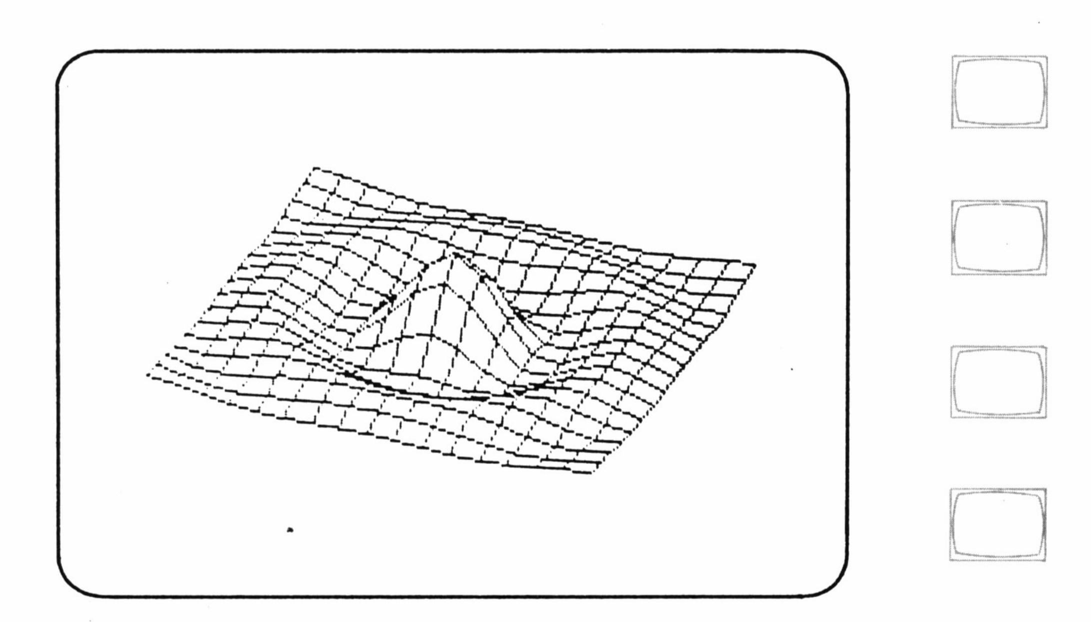


Figura 10.3

Al ser imposible dibujar cada punto de la superficie, nos aproximaremos considerando un subconjunto de dichos puntos. Escogeremos aquéllos cuya coordenada x/z se sitúe en una cuadrícula; dicho de otro modo, si proyectamos ortogonalmente el dibujo mirándolo desde arriba, esto es, ignorando los valores y, los puntos formarán una rejilla rectangular. Esta rejilla está compuesta por NX por NZ rectángulos en el plano x/z. Las coordenadas x de los vértices están equiespaciadas y varían entre XB y XT (XB < XT) y las coordenadas z, igualmente equiespaciadas, varían entre ZB y ZT (ZB < ZT). Habrá, por tanto (NX + 1) × (NZ + 1) vértices (X, Z) en la cuadrícula, identificables por una pareja de enteros (i, j)

$$X = XT + i(XB - XT)/NX \quad 0 \leqslant i \leqslant NX$$

$$Z = ZT + j(ZB - ZT)/NZ \quad 0 \leqslant j \leqslant NZ$$

El punto equivalente sobre la superficie es (X, Y, Z) donde $Y = f(X, Z)$. Cada uno de los (NX + 1) × (NZ + 1) puntos generados de esta forma se une a sus cuatro vecinos inmediatos en la rejilla (es decir, aquellos con el mismo valor de x o z), a menos que se sitúe en el borde, en cuyo caso se une a tres, o, en el caso de las esquinas, únicamente a dos vecinos. Consideraremos que la cuadrícula está formada por NX + 1 conjuntos de segmentos diferentes, cada uno de ellos con un determinado valor de x, y los correspondientes NZ + 1 conjuntos de segmentos con z fija. Por ejemplo, el conjunto I-ésimo de x fija ($0 \leqslant I \leqslant NX$) está compuesto de NZ segmentos que unen pares de puntos equivalentes a los puntos de la cuadrícula (I, J), (I, J + 1), donde $0 \leqslant J \leqslant NZ - 1$.

La superficie puede estar ondulada, de manera que no será visible toda la cuadrícula desde un determinado ángulo de visión. Explicaremos un método muy simple para eliminar las líneas ocultas de casos como éste, comenzando el dibujo desde la parte anterior de la figura.

Para simplificar el algoritmo, supondremos que el ojo está colocado siempre en el cuadrante positivo (es decir, EX > 0 y EZ > 0), y que además está mirando al origen (DX = DY = DZ = 0). En el caso de que la función no sea simétrica, y deseemos observarla desde otro ángulo, simplemente deberemos cambiar el signo de x y/o z en la función. Transformaremos a continuación la superficie a su posición OBSERVADA.

En esta posición, en primer lugar proyectamos ortogonalmente el borde frontal, o sea, las dos líneas de la cuadrícula correspondientes a $x = XT$ y $z = ZT$. Esta operación es equivalente a dibujar el primer conjunto de segmentos de x fija y el primer conjunto de z fija (los conjuntos número cero). A continuación nos movemos desde la parte delantera hacia atrás en etapas NX. Para realizar la etapa I-ésima ($1 \leqslant I \leqslant NX$), dibujamos en primer lugar las partes visibles de uno de los segmentos de cada uno de los NZ conjuntos de z fija. A su vez, el J-ésimo segmento de los anteriores ($1 \leqslant J \leqslant NZ$) unirá los puntos equivalentes a los puntos de cuadrícula (I − 1, J) e (I, J). Es decir, unimos los puntos correspondientes al segmento (I − 1) de x fija con los del segmento I, comenzando en la línea cero de z fija y trabajando desde delante a atrás. A continuación dibujamos las partes visibles del conjunto I-ésimo de segmentos de x fija. Toda esta operación está programada en la subrutina "superficie" *(surface)*.

Aún no hemos explicado cómo distinguir cuáles son las partes visibles de las líneas: subrutina "dibujalíneas" *(drawlin)*. Definimos un vector D de 256 posiciones, una para cada columna de *pixels* en la pantalla. Dibujamos en primer lugar los conjuntos cero de x y z fijas, e introducimos los valores equivalentes al *pixel* de los puntos de dichas líneas en el vector D. Calculamos a continuación los valores fila/columna de cada *pixel* para un segmento determinado, y lo dibujamos si y sólo si el valor de la fila para una determinada columna es mayor que el valor que en ese momento se encuentra almacenado en D. En ese caso, el nuevo valor del *pixel* se introduce en la posición correspondiente del vector D, y se continúa el proceso. Este método nos proporciona un algoritmo de eliminación de líneas ocultas para superficies matemáticas basado en el orden en el cual trazamos dichas líneas. El algoritmo está restringido al cuadrante positivo y a figuras que no excedan el tamaño de la pantalla gráfica; si cambiamos de cuadrante o ampliamos excesivamente la escala incurriremos en errores.

Ejercicio 10.4

Cambie las funciones "f" utilizadas en este programa. Por ejemplo, use $f = 4 \times \times \mathrm{SIN}(t)$ donde $t = \sqrt{(x^2 + z^2)}$.

Ejercicio 10.5

El programa anterior no dibuja la parte inferior de la figura, que aparecería por debajo de las líneas cero de x/z si su valor fuese lo suficientemente pequeño. Modifique el listado 10.4 para permitir que un valor pequeño y/o negativo de la función "asome" por la parte inferior del dibujo. Defina otro vector E(256), que inicialmente contendrá la información sobre el borde más próximo, y, si se da el caso de que algún punto está por debajo de los considerados como bordes, se dibujen alterando el correspondiente valor en el vector E. ¿Podría utilizarse el mismo procedimiento para dibujar las líneas?

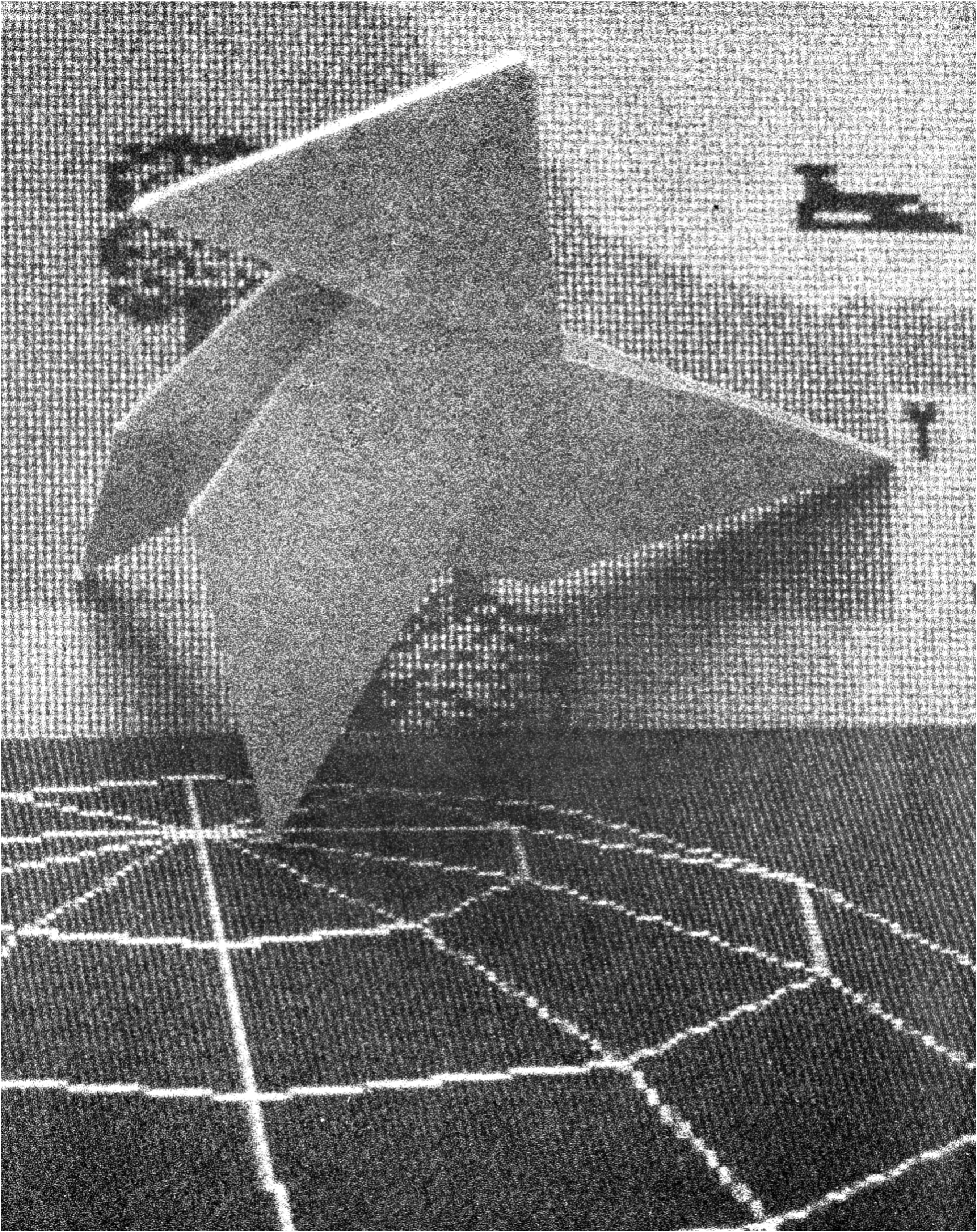
Programas completos

I. "rut3" y listado 10.1. Datos requeridos: las coordenadas de los vértices de un triángulo (X(I), Y(I), Z(I)), donde $1 \leqslant I \leqslant 3$. Pruebe con los valores (1, 0, 1), (1, 1, 0) y (0, 1, 1): introduzca a continuación los mismos vértices en orden diferente: (1, 1, 0), (1, 0, 1) y (0, 1, 1).

II. "rut1", "rut3" y los listados 9.3, "escena3" *(scene3)* y 10.2 "cubo" *(cube)*. Datos necesarios: HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ). Intente con 9, 6, (1, 2, 3), (0, 0, −1).

III. "rut1", "rut3" y los listados 9.10 "escena3" *(scene3)* y 10.3 "cuerpo de revolución" *(revbod)*. Datos necesarios: HORIZ, VERT, NUMH, NUMV ($\leqslant$ 15), PHI, (EX, EY, EZ), (DZ, DY, DZ). Intente 3.2, 2.2, 10, 10, 1, (1, 2, 3), (0, 0, −1).

IV. "rut1", "rut3" y el listado 10.4: "escena3", "superficie" "dibujalíneas" y "f" *(scene3, surface, drawlin* y *f)*. Datos necesarios: HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ), NX, XB, XT, NZ, ZB, ZT. Pruebe los valores 30, 20, (1, 2, 3), (0, 0, 0), 16, −18, 8, 16, −8, 8.



11
Proyecciones en perspectiva

Como hemos visto, la proyección ortogonal posee la propiedad de que líneas que eran paralelas en el espacio tridimensional se proyectan como líneas paralelas en el plano de visión. Aunque resulta muy útil, ciertamente esta proyección se nos antoja algo extraña: nuestro cerebro utiliza el fenómeno de la perspectiva del espacio tridimensional, y por ello intenta interpretar las figuras ortogonales como si se hubiesen realizado en perspectiva. Por ejemplo, los cubos de las figuras 9.1 y 10.1 aparecen distorsionados.

Resulta esencial, por tanto, producir una proyección que muestre fenómenos de perspectiva (es decir, que permita que las líneas paralelas se unan en el horizonte); de este modo, la parte más alejada de un objeto deberá verse de menor tamaño que la parte más cercana al observador. Por desgracia, no podremos echar mano de los métodos de *dibujo* descubiertos y utilizados por los artistas desde hace siglos; por el contrario, la geometría cartesiana en tres dimensiones y el concepto, ya introducido, de posición ACTUAL y OBSERVADA, nos permitirán encontrar una técnica bastante directa para cumplir este requisito.

Concepto de visión en perspectiva

Adquiriremos una idea más clara de este concepto si lo dividimos en dos partes: en primer lugar veremos una definición muy simple de lo que entendemos por visión. Imaginemos que cada punto visible del espacio envía un rayo de luz hasta el ojo. Naturalmente, el ojo no puede ver todo el espacio, sino que está limitado al cono de rayos que se proyecta sobre la retina, llamado *cono de visión*. En la figura 11.1 se muestra el cono de visión marcado con líneas de puntos. El eje de dicho cono

se denomina *rayo frontal.* Si imaginamos que el espacio se ha transformado en la posición OBSERVADA, con el eje situado en el origen, tendremos el rayo frontal colocado en el eje z positivo.

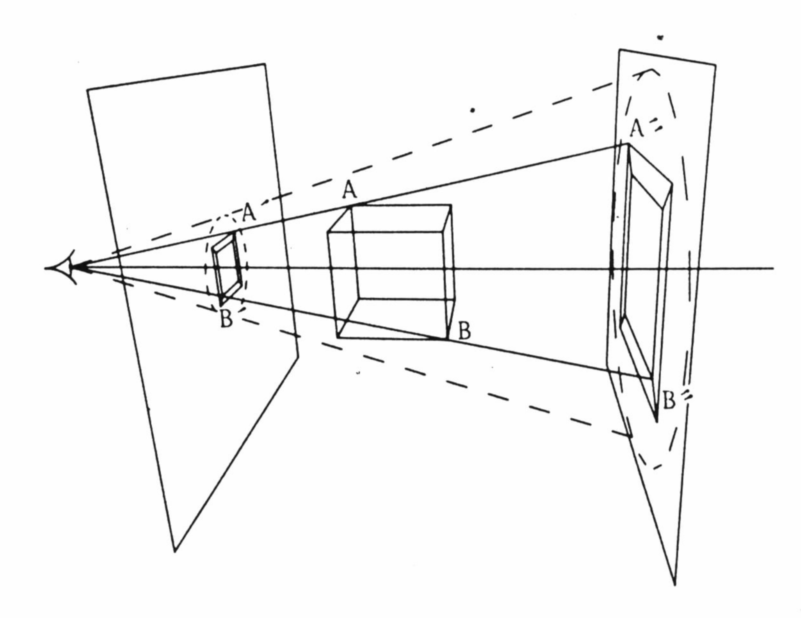


Figura 11.1

Situaremos el plano de visión (aquí llamado *plano de perspectiva*) perpendicular al eje del cono de visión, y situado a una distancia D del ojo. Para producir la proyección en perspectiva, marcamos los puntos de intersección de cada rayo con dicho plano. Por supuesto hay infinitos rayos, y esta tarea parece imposible: en realidad el problema no es tan grande, ya que consideramos únicamente aquellos rayos que proceden de partes importantes de la figura; en concreto, los vértices y puntos finales de segmentos. El dibujo final será el resultado de unir estos puntos proyectados de la misma manera como estaban relacionados en el espacio tridimensional, y a continuación asociar el plano de perspectiva con la pantalla gráfica del ordenador.

En la figura 11.1 se muestra un cubo observado por un ojo y proyectado sobre dos planos, uno anterior y otro posterior; la escena completa, además, está dibujada en perspectiva. Con línea continua se han marcado dos rayos como ejemplo: el primero parte del ojo hasta A, uno de los vértices de la cara más proxima del cubo (con respecto al ojo), y el segundo a B, uno de los vértices de la cara posterior. Las proyecciones en perspectiva de estos puntos en el plano cercano al ojo son A′ y B′, mientras que en el otro plano son A″ y B″. Obsérvese que ambas proyecciones tienen la misma forma y orientación, difiriendo únicamente en tamaño.

Cálculo de la proyección en perspectiva de un punto

Supongamos que el plano de perspectiva se sitúa a una distancia d del ojo (la variable PPD en programas posteriores). Considérese un punto P $\equiv (x, y, z)$ en el espacio que envía un rayo hacia el ojo. Debemos calcular el punto en que dicho rayo corta al plano de visión (el plano $z = d$); supongamos que dicho punto es P$' \equiv (x', y', d)$. Calcularemos en primer lugar el valor de y' refiriéndonos a la figura 11.2. Por triángulos semejantes se puede demostrar que $y'/d = y/z$, es decir, $y' = y \times d/z$. De la misma forma, $x' = x \times d/z$. Por tanto, P$' \equiv (x \times d/z, y \times d/z, d)$. Cuando asociamos el plano de visión con el sistema de coordenadas x/y de la pantalla gráfica, podemos despreciar la coordenada $z = d$.

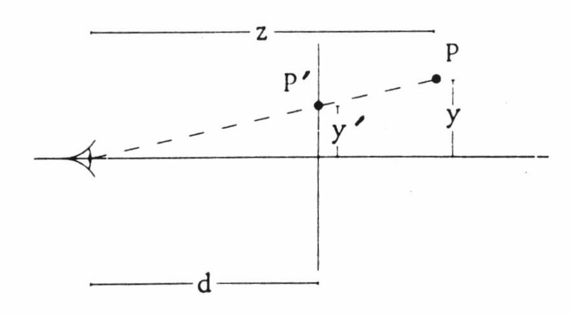


Figura 11.9

Ejemplo 11.1

Calcúlese la proyección en perspectiva de un cubo con ocho vértices $(0, 0, 4) + (\pm 1, \pm 1, \pm 1)$ en el plano de perspectiva $z = 4$, situando el ojo en el origen y el rayo frontal en el eje z positivo.

El espacio está definido de manera que la escena está colocada en posición OBSERVADA. Podemos calcular las proyecciones de los ocho vértices utilizando el método anterior. Por ejemplo, el punto (1, 1, 3) se proyecta en $(1 \times 4/3, 1 \times 4/3, 4) = (4/3, 4/3, 4) \rightarrow (4/3, 4/3)$ en la pantalla. De la misma forma obtenemos las ocho proyecciones

$$
\begin{array}{llll}
(1, 1, 3) & \rightarrow (4/3, 4/3) & (1, -1, 3) & \rightarrow (4/3, -4/3) \\
(-1, 1, 3) & \rightarrow (-4/3, 4/3) & (-1, -1, 3) & \rightarrow (-4/3, -4/3) \\
(1, 1, 5) & \rightarrow (4/5, 4/5) & (1, -1, 5) & \rightarrow (4/5, -4/5) \\
(-1, 1, 5) & \rightarrow (-4/5, 4/5) & (-1, -1, 5) & \rightarrow (-4/5, -4/5)
\end{array}
$$

resultando el diagrama que se muestra en la figura 11.3a.

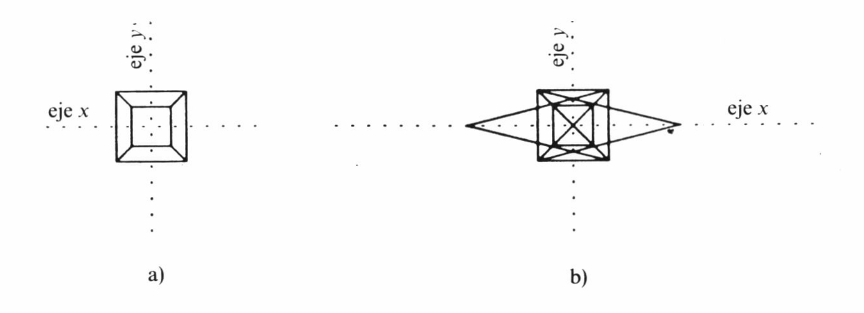


Figura 11.3

Propiedades de la transformación en perspectiva

1) La transformación en perspectiva de una línea recta, por ejemplo Γ_3, es una línea recta, por ejemplo Γ_2. Esto es obvio debido a que el origen (el ojo) y la línea Γ_3 forman un plano (que llamaremos Ω) en el espacio tridimensional, y todos los rayos que se emiten desde puntos de Γ_3 pertenecen a dicho plano. (Si la recta penetra en el ojo, Ω degenera en una línea). Evidentemente, Ω corta al plano de perspectiva en una línea Γ_2 (o degenera en un punto) de manera que la proyección en perspectiva de un punto de la línea original Γ_3 debe pertenecer ahora a la nueva línea Γ_2. Es importante observar que una línea recta no se curva en una proyección en perspectiva.

2) La transformación en perspectiva de una cara (secuencia cerrada de segmentos rectilíneos coplanares) es una cara en el plano de perspectiva. Si la cara es un área rodeada por n segmentos coplanares, su transformada es otra área en el plano $z = d$ rodeada por las transformadas de los n segmentos. Insistimos de nuevo que no aparecen curvas en esta proyección: si así fuera, la tarea de realizar dibujos en perspectiva sería bastante más complicada.

3) La proyección de una cara convexa es también convexa. Supongamos que la proyección de la cara F_1 es la cara F_2. Al ser la proyección de una cara cerrada también cerrada, y las rectas rectas también, los puntos que pertenecían a F_1, estarán ahora contenidos en F_2. Supongamos que F_2 no fuera convexa: existirían entonces dos puntos $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$ dentro de F_2 tales que una recta que los uniera no pertenecería en su totalidad a la cara. De ahí se deduce que hay al menos un punto $\boldsymbol{p}$ de la recta en el exterior de F_2. Si $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$ son proyecciones de los puntos $\boldsymbol{q}_1$ y $\boldsymbol{q}_2$ de F_1, $\boldsymbol{p}$ será la proyección de un punto $\boldsymbol{q}$ del segmento que une $\boldsymbol{q}_1$ y $\boldsymbol{q}_2$. Al ser F_1 convexo, $\boldsymbol{q}$ deberá pertenecer a F_1, y por consiguiente $\boldsymbol{p}$ deberá estar incluido en F_2: ha aparecido una contradicción, por lo que nuestra hipótesis queda demostrada.

4) Cualquier número de líneas paralelas infinitas parecen unirse en un punto, llamado *punto de fuga*. Si tomamos una línea cualquiera (con vector de base $\boldsymbol{p}$) de un conjunto de líneas paralelas con vector de dirección $\boldsymbol{h}$

$$\boldsymbol{p} + \mu\boldsymbol{h} \equiv (x_p, y_p, z_p) + \mu(x_h, y_h, z_h)$$

donde $z_h > 0$; por tanto, la transformación en perspectiva de un punto cualquiera de la línea será

$$\left(\frac{(x_p + \mu x_h) \times d}{(z_p + \mu z_h)}, \frac{(y_p + \mu y_h) \times d}{(z_p + \mu z_h)}\right)$$

que se puede reescribir como

$$\left(\frac{(x_h + x_p/\mu) \times d}{(z_h + z_p/\mu)}, \frac{(y_h + y_p/\mu) \times d}{(z_h + z_p/\mu)}\right)$$

Según nos movemos hacia coordenadas z mayores (es decir, según μ tiende a infinito), la línea se mueve hacia el punto de fuga, el cual viene dado por $(d \times x_h/z_h, d \times y_h/z_h)$. Este punto de fuga es independiente de $\boldsymbol{p}$, vector de base de la recta, y por tanto todas las líneas paralelas en la dirección $\boldsymbol{h}$ convergerán en el mismo punto. Por supuesto, despreciamos el caso $z_h < 0$, porque la línea desaparecería fuera del cono de visión al tender $\mu \to \infty$.

5) Los puntos de fuga de todas las líneas colocadas en planos paralelos son colineales. Supóngase que tenemos un conjunto de planos paralelos con una normal $\boldsymbol{n} \equiv (x_n, y_n, z_n)$. Si una línea cualquiera de uno de estos planos tiene la dirección $\boldsymbol{h} \equiv (x_h, y_h, z_h)$, entonces $\boldsymbol{h}$ será perpendicular a $\boldsymbol{n}$ (todas las líneas situadas en un plano son perpendiculares a su normal). Por consiguiente el producto escalar $\boldsymbol{n} \cdot \boldsymbol{h} = 0$, que expresado en forma de coordenadas es

$$x_n \times x_h + y_n \times y_h + z_n \times z_h = 0$$

dividiendo por z_h obtenemos

$$x_n \times x_h/z_h + y_n \times y_h/z_h + z_n = 0$$

y el punto en cuestión $(d \times x_h/z_h, d \times y_h/z_h)$ se sitúa sobre la recta

$$x_n \times x + y_n \times y + d \times z_n = 0$$

como se quería demostrar.

Ejemplo 11.2

Encuéntrense los puntos de fuga de los lados del cubo del ejemplo 11.1, y los de las diagonales de sus caras superior e inferior.

Dividimos las doce aristas del cubo en tres conjuntos de cuatro aristas, paralelos respectivamente a los ejes x, y y z, y por tanto con vectores de dirección $(1, 0, 0)$, $(0, 1, 0)$ y $(0, 0, 1)$. Los dos primeros conjuntos tienen un valor z cero, y por tanto sus prolongaciones escapan del cono de visión y serán ignoradas. La tercera dirección presenta un punto de fuga $(4 \times 0/1, 4 \times 0/1) \equiv (0, 0)$ en el plano de visión. Las diagonales de las caras superior e inferior tienen direcciones $(1, 0, 1)$ y $(-1, 0, 1)$ respectivamente la diagonal mayor y la diagonal menor. La diagonal mayor del plano superior es $(-1, 1, 3) + \mu(1, 0, 1)$, y su punto de fuga será $(4 \times 1/1, 4 \times 0/1) \equiv (4, 0)$. La diagonal menor del plano superior es $(1, 1, 3) + \mu(-1, 0, 1)$ y su punto correspondiente será $(4 \times -1/1, 4 \times 0/1) \equiv (-4, 0)$. Un cálculo semejante nos conduce a situar los puntos de fuga de las diagonales mayor y menor de la cara inferior también en $(4, 0)$ y $(-4, 0)$ respectivamente. En la figura 11.3b hemos prolongado las aristas calculadas hasta sus puntos de fuga. Obsérvese que todas las líneas mencionadas se sitúan en dos planos paralelos (las caras superior e inferior del cubo) y por tanto los puntos de fuga son colineales: comprobamos fácilmente que $(4, 0)$, $(0, 0)$ y $(-4, 0)$ pertenecen todos ellos al eje x, estando obviamente en línea recta. De igual forma se puede demostrar que los puntos de fuga de las diagonales de las caras laterales se sitúan en una línea vertical que pasa por el origen.

Ejercicio 11.1

Dibuje una visión en perspectiva de un tetraedro con vértices $(1, 1, 5)$, $(1, -1, 3)$, $(-1, 1, 3)$ y $(-1, -1, 5)$. Búsquense los puntos de fuga (dentro del cono de visión) de las rectas que unen pares de puntos medios de las aristas del tetraedro.

Programación de la transformación en perspectiva

El programa principal para dibujos en perspectiva de cualquier tipo de escena es el mismo que el utilizado en proyección ortogonal, es decir, el listado 9.2. Al igual que antes, la escena global se crea por una llamada a una subrutina "escena3" semejante a las mostradas en el capítulo 9. A menudo tendremos que calcular explícitamente la matriz de paso ACTUAL a OBSERVADA, de tal forma que el ojo esté en posición OBSERVADA en el origen mirando hacia el eje z positivo. Esta misión está asignada a la subrutina "observación3" dada en el capítulo 9 (listado 9.1). Se realizan llamadas a subrutinas de construcción, teniendo cada una una matriz R como parámetro. Finalmente dibujamos la figura dentro de la propia subrutina de construcción o utilizando una subrutina "dibujo".

La única diferencia entre el programa que dibuja en perspectiva y el programa del capítulo 9 (proyección ortogonal), está en el cálculo de las coordenadas de la imagen proyectada en el plano de visión. A diferencia del caso ortogonal, en la

proyección en perspectiva las coordenadas del plano de visión no pueden identificarse con los valores x e y del punto en posición OBSERVADA. Debemos almacenar la transformación de perspectiva de los vértices en los vectores V y W: el vértice I-ésimo (X(I), Y(I), Z(I)) en posición OBSERVADA se proyecta a (V(I), W(I)). Los valores de los vectores V y W vienen dados por

$$V(I) = X(I)*PPD/Z(I) \quad y \quad W(I) = Y(I)*PPD/Z(I) \quad \text{para } I = 1, 2, ..., NOV$$

aquí se le asigna a PPD el valor 3*VERT, por las razones que veremos en la siguiente sección. El cálculo de V y W se puede realizar en la subrutina de construcción, en "escena3" o en la subrutina "dibujo"; la elección depende simplemente de la figura que se esté considerando.

Ejemplo 11.3

Dibuje una escena fija (los dos cubos descritos en el ejemplo 9.2 en perspectiva desde distintos puntos de observación, haciendo HORIZ = 9 y VERT = 6. Los subrutina "escena3" que se necesita viene dada en el listado 11.1; obsérvese que esta subrutina calcula PPD (compárese con el listado 9.6). El listado sitúa al grupo de cubos en su posición actual utilizando la subrutina "cubo" del listado 9.7, y a continuación realiza un bucle con diferentes posiciones de OBSERVADOR. Para cada nueva posición se llama a "observación3", que necesita los datos (EX, EY, EZ) y (DX, DY, DZ) para calcular la matriz de paso ACTUAL a OBSERVADOR. A continuación se llama a la subrutina "dibujo" en perspectiva (listado 11.2). Dicha subrutina utiliza la matriz para transformar los vértices desde su posición ACTUAL (almacenada) a la posición OBSERVADA y sitúa las coordenadas de los vértices proyectados en los vectores V y W, de acuerdo con las ecuaciones anteriores. Finalmente, la subrutina dibuja las aristas de los cubos en perspectiva.

La figura 11.4 ha sido realizada utilizando (EX, EY, EZ) ≡ (15, 10, 5) y (DX, DY, DZ) ≡ (0, 0, 0). Compárese esta figura con la proyección ortogonal de la misma escena presentada en la figura 9.2.

Listado 11.1

```
6000 REM escena3/figura 9.2(vari
as vistas)
6010 DIM X(16): DIM Y(16): DIM Z
(16)
6020 DIM V(16): DIM W(16): DIM L
(2,24)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4)
```

```
6040 LET cube=6500: LET drawit=7
000
6050 LET PPD=3*VERT: LET NOV=0:
LET NOL=0
6059 REM Situa dos cubos en la p
osicion ACTUAL
6060 GO SUB idR3
6070 GO SUB cube
6080 LET TX=3: LET TY=1.5: LET T
Z=2: GO SUB tran3: GO SUB mult3
6090 GO SUB cube
6099 REM Bucle de varias vistas
6100 GO SUB idR3: GO SUB look3
6110 CLS : GO SUB drawit
6120 GO TO 6100
6130 RETURN
```

Listado 11.2

```
7000 REM dibujo/perspectiva
7001 REM Datos de entrada PPD,NO
V,NOL,X(NOV),Y(NOV),Z(NOV),L(2,N
OL),R(4,4)
7009 REM Situa los vertices en l
a posicion observada
7010 FOR I=1 TO NOV
7020 LET XX=X(I)*R(1,1)+Y(I)*R(1
,2)+Z(I)*R(1,3)+R(1,4)
7030 LET YY=X(I)*R(2,1)+Y(I)*R(2
,2)+Z(I)*R(2,3)+R(2,4)
7040 LET ZZ=X(I)*R(3,1)+Y(I)*R(3
,2)+Z(I)*R(3,3)+R(3,4)
7049 REM Proyeccion en perspecti
va
7050 LET V(I)=XX*PPD/ZZ
7060 LET W(I)=YY*PPD/ZZ
7070 NEXT I
7079 REM Dibuja las lineas
7080 FOR I=1 TO NOL
7090 LET L1=L(1,I): LET L2=L(2,I
)
7100 LET XPT=V(L1): LET YPT=W(L1
```

```
): GO SUB moveto
7110 LET XPT=V(L2): LET YPT=W(L2
): GO SUB lineto
7120 NEXT I
7130 RETURN
```

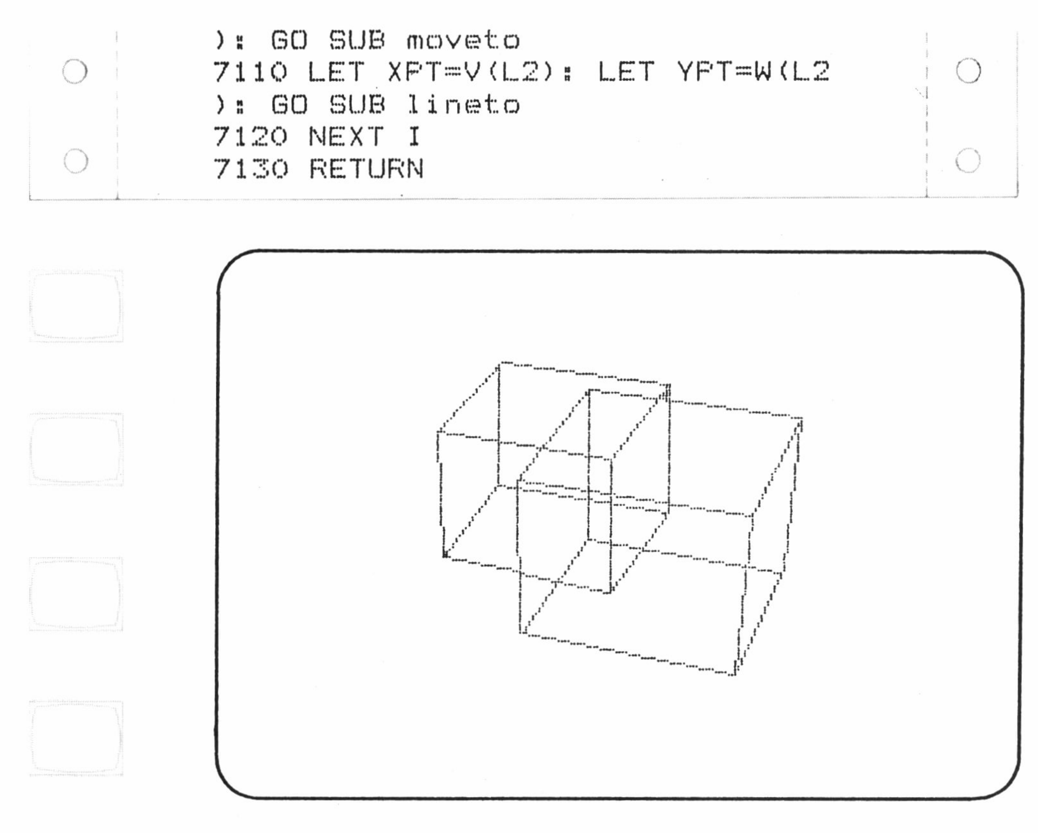


Figura 11.4

Ejercicio 11.2

Dibuje varias perspectivas de un "esqueleto" de tetraedro y de pirámide.

Elección del plano de perspectiva

El único valor que se necesita para la transformación en perspectiva, que todavía no hemos estudiado, es PPD, la distancia del plano de perspectiva al ojo. Si observamos la figura 11.1, se advierte que un valor diferente de PPD produce el mismo dibujo aunque de diferente tamaño. ¿Cuál escogemos? ¿Existe un valor correcto?

Si consideramos una situación práctica, en la que el observador está situado enfrente de la pantalla de televisión y el plano de perspectiva se identifica con el plano de la pantalla, podemos utilizar una regla empírica, según la cual el observador se sitúa a una distancia que es aproximadamente tres veces la altura de la pantalla. Traduciendo esta distancia a *pixels,* resulta una distancia 3*VERT (valor utilizado anteriormente). Si utilizamos valores mayores de PPD estamos ampliando la imagen, mientras que si PPD es menor que 3*VERT damos una impresión de *lejanía.*

Recorte

En principio, los objetos pueden estar colocados en cualquier lugar del espacio, incluso detrás del ojo, aunque en nuestro caso se consideran tan sólo aquellas coordenadas z positivas en la posición OBSERVADA. Aun así, bastantes puntos estarán fuera del cono de visión, y por tanto fuera de la pantalla. De hecho, incluso una parte del cono de visión está fuera de la pantalla (después de todo, podemos ver la parte exterior del área gráfica). Así pues, deberemos seleccionar un subconjunto del cono de visión que llamaremos *pirámide de visión*. Todos los puntos que caigan fuera de esta pirámide (es decir, aquellos cuya perspectiva los transforman en puntos fuera de la pantalla) deben ser ignorados. Ya anotamos que el Spectrum presenta un mensaje de error si se intenta dibujar una línea a un punto situado fuera del área gráfica. Por consiguiente es esencial que utilicemos la versión "recorte" de la subrutina "trazalínea" (listado 3.3) con el fin de evitar problemas. De hecho, limitaremos aún más las escenas, de manera que todos los vértices en posición OBSERVADA presenten valores positivos en z; esto es, todos los objetos deberán situarse en la parte frontal del ojo (aunque no necesariamente dentro del cono de visión). Así se evita que en una proyección en perspectiva particular encontremos puntos situados detrás del ojo apareciendo en la pantalla.

Ejercicio 11.3

Experimente con proyecciones en perspectiva de todo tipo de figuras de líneas: por ejemplo, cuerpos de revolución, poliedros regulares. Considérense casos en los que el objeto se dibuja dentro de la propia subrutina de construcción, es decir, aquellos en que los valores de V y W deben calcularse ahí y no en la subrutina "dibujo". Modifique el programa que dibuja el reactor de la figura 9.3 para obtenerlo en perspectiva; observe que conforme el ojo se aleja, el aeroplano se va volviendo más pequeño, fenómeno que no se daba en la proyección ortogonal.

Ejercicio 11.4

Escriba un algoritmo de líneas ocultas para un cuerpo convexo, similar al presentado en el capítulo 10.

Nótese que al ser la proyección de una cara convexa otro polígono convexo en el plano de visión, necesitamos únicamente calcular las coordenadas de los vértices de la cara proyectada, y calcular si la cara está situada en sentido antihorario (en cuyo caso la dibujaremos) u horario (en cuyo caso se debe ignorar).

Ejercicio 11.5

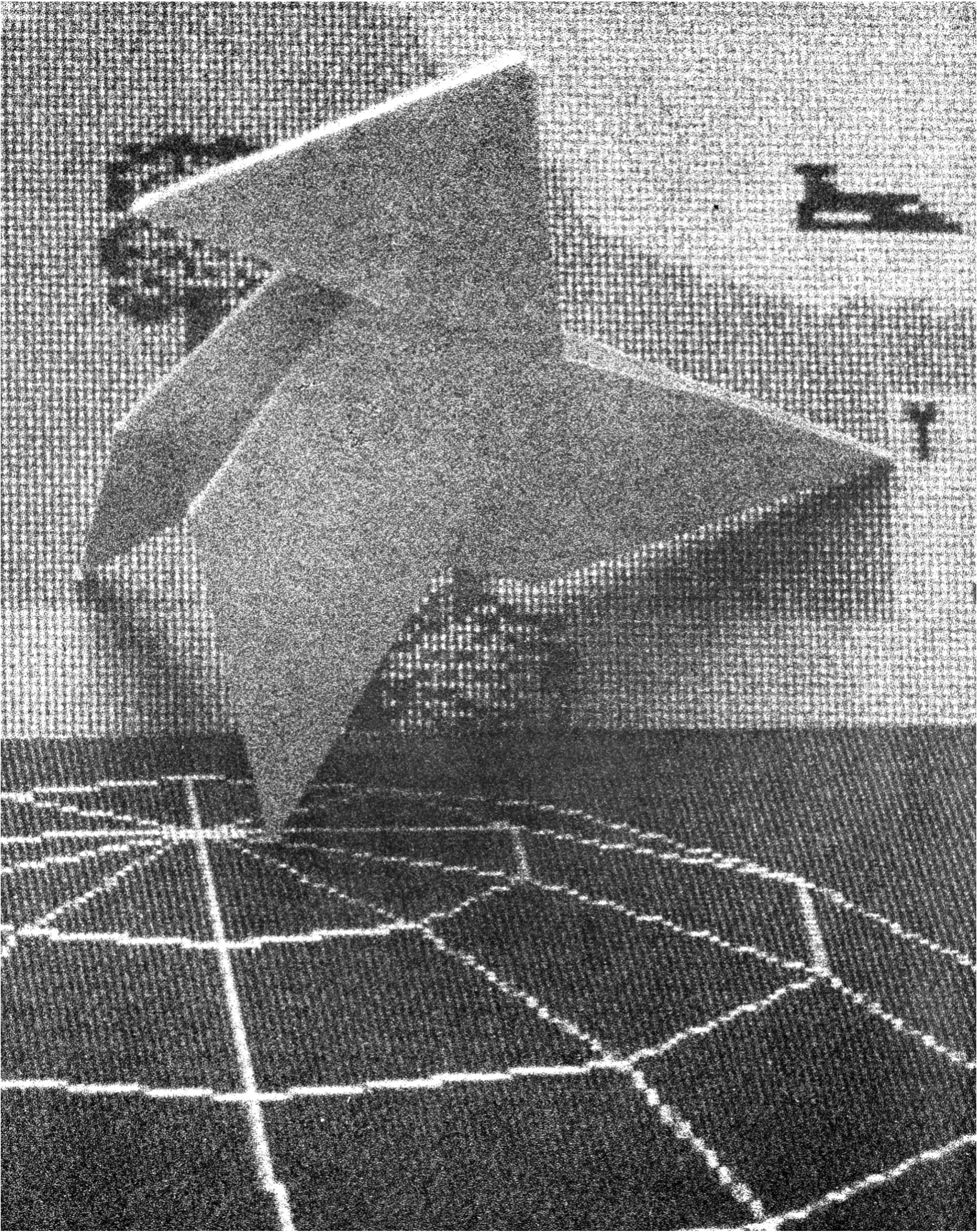
Escriba un programa que dibuje una visión en perspectiva de una superficie matemática, semejante a la mostrada en el capítulo 10. El método será exactamente

equivalente al mostrado en el listado 10.4, con la única excepción de que deberá trabajar con los vectores V/W en lugar de los vectores X/Y.

Estos algoritmos de líneas y superficies ocultas se adecúan perfectamente a objetos únicos definidos especialmente, dentro de unas ciertas condiciones. Consideremos ahora el caso más general en el que hay varios objetos repartidos en el espacio.

Programas completos

I. "rut1", "rut3" y los listados 9.4 "cubo" *(cube),* 11.1 "escena3" *(scene3)* y 11.2 "dibujo" *(drawit).* Datos necesarios: HORIZ, VERT y valores repetidos para (EX, EY, EZ) y (DX, DY, DZ). Intente 9, 6 (5, 15, 10) y (0, 0, 0); (1, 2, 20) y (0, 0, 1).



12
Un algoritmo de líneas ocultas de propósito general

Como hemos visto en los capítulos anteriores, asumimos que los objetos son ubicados por la subrutina “escena3”; debemos recordar ahora que los NOV vértices de la escena se almacenan en los vectores X, Y y Z, y sus proyecciones en perspectiva en el plano de visión, en los vectores V y W. Las NOV caras quedan almacenadas como una lista de índices de vértices (hasta un máximo de seis) en la matriz F, y el número de aristas de cada cara, en el vector H.

Suponemos también que todos los objetos son cerrados. No se requiere que todos ellos sean convexos, pero su superficie debe estar compuesta de caras convexas almacenadas en orientación antihoraria. De este modo es imposible ver el otro lado de una cara cualquiera; es decir, cuando la figura se proyecta en el plano de visión, vemos solamente aquellas caras que mantienen su orientación antihoraria. Estrictamente hablando, esto significa que no podemos dibujar objetos planos. En el caso de necesitar alguno en una determinada escena, podemos evitar el problema almacenando cada cara del objeto plano dos veces: una de ellas en sentido horario y otra antihorario; de esta forma, independientemente de la posición del ojo, obtendremos una proyección en perspectiva de una y sólo una de estas caras. También asumimos que todas las líneas de la escena son aristas de dos caras contiguas: así pues, una única línea deberá almacenarse como un polígono degenerado de dos lados. Todas estas restricciones vienen aconsejadas con el fin de dar una mayor velocidad al algoritmo de líneas ocultas. La velocidad es un factor realmente limitativo en nuestro caso, ya que estamos aproximándonos a los límites del poder de procesado de nuestro Spectrum. Incluso un dibujo sencillo, como los dos cubos de la figura 12.1, llevan más de cinco minutos de ejecución; la dos estrellas de la figura 12.2 tardan por encima de los treinta minutos en realizarse. El Spectrum no ha sido diseñado para ejecutar algoritmos tan complejos, por lo que es de alabar la labor del equipo de

diseño de Sinclair, que ha conseguido que el Spectrum, a pesar de todo, alcance tan excelentes resultados.

En cualquier caso, creemos importante estudiar algoritmos generales de líneas ocultas por razones puramente educativas. Pensamos que es esencial para cualquiera que posea un interés mayor que un simple pasar el rato, que comprenda los problemas subyacentes en el dibujo de objetos tridimensionales con supresión de líneas ocultas. La subrutina del listado 12.1 es un algoritmo de líneas ocultas general, que puede ser transferido a ordenadores mayores, donde se ejecutará fácilmente. Si usted tiene acceso a algún ordenador más potente, sería muy instructivo que intentase ejecutar nuestros programas en él.

Si deseamos producir un dibujo con líneas ocultas de una escena almacenada en posición OBSERVADA, deberemos comenzar por comparar cada línea de los objetos con cada cara. Debido a las restricciones anteriores, necesitaremos comparar las líneas únicamente con las caras visibles; es decir, con aquellas que, al proyectarse, mantienen su orientación antihoraria.

Supongamos que una línea cualquiera Γ_3 en posición OBSERVADA une los puntos (x_1', y_1', z_1') y (x_2', y_2', z_2'), de manera que un punto cualquiera de la línea es

$$(1 - \phi)(x_1', y_1', z_1') + \phi(x_2', y_2', z_2')$$

A la proyección de los dos puntos terminales de la línea los llamaremos (x_1, y_1) y (x_2, y_2), situados, obviamente, en el plano de visión en perspectiva. De este modo, Γ_3 se proyectará en dicho plano como una línea Γ_2 en la que un punto cualquiera será

$$(1 - \mu)(x_1, y_1) + \mu(x_2, y_2)$$

Obsérvese que el punto $(1 - \phi)(x_1', y_1', z_1') + \phi(x_2', y_2', z_2')$ no se transforma necesariamente en el punto $(1 - \phi)(x_1, y_1) + \phi(x_2, y_2)$: es decir, ϕ no tiene por qué ser igual a μ.

Hagamos que una cara cualquiera Ω_3 se proyecte en un área Ω_2 en el plano de visión y supongamos que los H vértices de esta cara proyectada son $(\bar{x}_i, \bar{y}_i)$, donde $1 \leqslant i \leqslant$ H. Supongamos que la arista i-ésima de Ω_2 corta a Γ_2 en $(1 - \lambda_i)$ $(\bar{x}_i, \bar{y}_i) + \lambda_i(\bar{x}_{i+1}, \bar{y}_{i+1})$. El valor de λ_i nos indica el lugar de la intersección: si $\lambda_i < 0$ ó $\lambda_i > 1$, entonces Γ_2 intersecta a la arista i-ésima en un punto exterior al área Ω_2; si $0 \leqslant \lambda_i \leqslant 1$, entonces Γ_2 atraviesa el área Ω_2 en un punto de la arista i-ésima. Al ser convexa la proyección de una cara convexa, el número de puntos de corte será, bien cero (y consecuentemente no hay intersección) o dos (quizá coincidentes). Sólo es necesario tener en cuenta el caso de dos puntos no coincidentes: llamaremos a sus valores μ(en Γ_2) $\mu_{\text{mín}}$ y $\mu_{\text{máx}}$, donde $\mu_{\text{mín}} < \mu_{\text{máx}}$. Así pues, los puntos de intersección de Γ_2 serán $(1 - \mu_{\text{mín}})$ $(x_1, y_1) + \mu_{\text{mín}}(x_2, y_2)$ y $(1 - \mu_{\text{máx}})$ $(x_1, y_1) + \mu_{\text{máx}}(x_2, y_2)$.

El siguiente paso es decidir si el subsegmento de Γ_2 comprendido entre estos dos puntos es visible o no. Esta tarea se realiza encontrando el punto medio del segmento, $(x_{\text{mid}}, y_{\text{mid}}) = (1 - \mu_{\text{mid}})$ $(x_1, y_1) + \mu_{\text{mid}}(x_2, y_2)$, donde $\mu_{\text{mid}} = (\mu_{\text{mín}} + \mu_{\text{máx}})/2$. A con-

tinuación localizamos el punto $(\hat{x}, \hat{y}, \hat{z})$ de Γ_3 cuya proyección en el plano de perspectiva es (x_{mid}, y_{mid}). El segmento de la línea comprendida entre los puntos con valores $\mu_{mín}$ y $\mu_{máx}$ será oculto si y sólo si (x, y, z) y el ojo están en lados opuestos de un plano infinito que contenga a Ω_3. La ecuación del plano se puede calcular utilizando los métodos descritos en el capítulo 7, y utilizaremos la representación del plano como función para comprobar la condición anterior.

Obsérvese que $\hat{x}$*PPD/$z = x_{mid}$, $\hat{y}$*PPD/$z = y_{mid}$, y, además $(\hat{x}, \hat{y}, \hat{z})$ se sitúa en Γ_3. Por tanto, para algún valor de ϕ se cumple que

$$x = (1 - \phi)x_1' + \phi x_2', \quad y = (1 - \phi)y_1' + \phi y_2', \quad \text{y} \quad z = (1 - \phi)z_1' + \phi z_2'$$

Así pues,

$$x_{mid} = \frac{(x_1' + \phi(x_2' - x_1'))*\text{PPD}}{z_1' + \phi(z_2' - z_1')}$$

y

$$y_{mid} = \frac{(y_1' + \phi(y_2' - y_1'))*\text{PPD}}{z_1' + \phi(z_2' - z_1')}$$

es decir,

$$\phi = \frac{x_{mid}*z_1' - x_1'*\text{PPD}}{(x_2' - x_1')*\text{PPD} - x_{mid}*(z_2' - z_1')}$$

$$= \frac{y_{mid}*z_1' - y_1'*\text{PPD}}{(y_2' - y_1')*\text{PPD} - y_{mid}*(z_2' - z_1')}$$

De este modo se puede calcular ϕ, y por tanto (x, y, z) el cual a su vez se utiliza para deducir si el subsegmento Γ_2 es visible o no.

El algoritmo dado en el listado 12.1 compara cada arista de los objetos con todas las caras visibles (antihorarias). Obsérvese que se consideran sólidos todos los objetos, es decir, no aparecen caras planas independientes de manera que pueda verse su parte posterior (orientación horaria). Las aristas están implícitamente almacenadas en los datos de las caras, y cada una de ellas aparece dos veces, una desde el vértice IV1 al vértice IV2 (por ejemplo) y otra desde el vértice IV2 al IV1. En lugar de duplicar el trabajo, consideramos sólo el caso en que IV1 < IV2. Además comparamos las líneas únicamente con caras visibles, porque si la línea está tapada por una cara oculta (una cara con orientación horaria) la parte invisible de dicha línea debe estar tapada asimismo por caras antihorarias.

Supongamos que estamos comparando la línea Γ_2, que une los vértices IV1 e IV2 con la cara K-ésima; hemos calculado NRL subsegmentos visibles de la línea: suponemos que NRL es menor de 50. Los valores μ de los puntos terminales del segmento M-ésimo visible se almacenan en la matriz L como L(1, M) y L(2, M).

Inicialmente NRL = 1, L(1, 1) = 0 y L(2, 1) = 1; es decir, se supone que la línea completa es visible. En este momento, si se descubre una parte oculta del segmento, especificada por los valores $\mu_{mín}$ y $\mu_{máx}$ (las variables MIN y MAX), se modifican los valores de la matriz L y de NRL de acuerdo con la nueva situación.

Una vez que la línea ha sido comparada con todas las caras visibles dispondremos de NRL segmentos visibles que pueden a continuación dibujarse en pantalla. Si en un momento dado NRL se hace cero, significa que la línea está totalmente tapada, y no hay necesidad de continuar realizando comparaciones con las demás caras.

Listado 12.1

```
7000 REM algoritmo general de li
neas ocultas
7001 REM Datos de entrada NOV,NO
L,X(NOV),Y(NOV),Z(NOV),V(NOV),W(
NOV),
7002 REM F(6,NOF),H(NOF),PPD,R(4
,4)
7010 DIM L(2,50): DIM G(NOF): LE
T EPS=0.000001
7018 REM Comprueba la orientacio
n de la cara Iesima proyectada:
7019 REM Si es antihoraria G(I)=
1; si es horaria o degenerada G(
I)=0
7020 FOR I=1 TO NOF
7030 LET I1=F(1,I): LET X1=V(I1)
: LET Y1=W(I1)
7040 LET I2=F(2,I): LET X2=V(I2)
: LET Y2=W(I2)
7050 LET I3=F(3,I): LET X3=V(I3)
: LET Y3=W(I3)
7060 LET DX1=X2-X1: LET DY1=Y2-Y
1
7070 LET DX2=X3-X2: LET DY2=Y3-Y
2
7080 LET G(I)=0
7090 IF DX1*DY2-DX2*DY1>0 THEN
LET G(I)=1
7100 NEXT I
7109 REM Encuentra la linea J en
 los bordes de la cara I
7110 FOR I=1 TO NOF
```

```
7120 LET HH=H(I): LET IV1=F(HH,I
)
7130 LET X1=V(IV1): LET Y1=W(IV1
)
7140 FOR J=1 TO HH
7150 LET IV2=F(J,I)
7160 LET X2=V(IV2): LET Y2=W(IV2
)
7170 IF IV1>IV2 THEN  GO TO 8160
7179 REM Inicializa las variable
s
7180 LET NRL=1: LET L(1,1)=0: LE
T L(2,1)=1
7190 LET CA=X2-X1: LET CB=Y1-Y2
7200 LET CC=-X1*CB-Y1*CA
7209 REM Compara esta linea con
la cara K
7210 FOR K=1 TO NOF
7220 IF G(K)=0 THEN  GO TO 8040
7230 IF K=I THEN  GO TO 8040
7240 LET IN=H(K)
7247 REM Bucle para encontrar do
s puntos de interseccion de la l
inea proyectada
7248 REM con la cara proyectada.
 Estos puntos estan especificado
s por los valores
7249 REM de MU MIN y MAX
7250 LET MAX=-1: LET MIN=2
7260 LET JV1=F(IN,K)
7270 LET VX1=V(JV1): LET WY1=W(J
V1)
7280 LET S1=SGN (CA*WY1+CB*VX1+C
C)
7290 FOR M=1 TO IN
7300 LET JV2=F(M,K)
7310 LET VX2=V(JV2): LET WY2=W(J
V2)
7320 LET S2=SGN (CA*WY2+CB*VX2+C
C)
7330 IF S1=S2 THEN  GO TO 7500
7340 LET XE=VX1-VX2: LET YE=WY1-
WY2
7350 LET XF=VX1-X1: LET YF=WY1-Y
1
```

```
7360 LET DISC=CA*YE+CB*XE
7369 REM Si la linea es paralela
 a la linea en la cara sale del
bucle de caras
7370 IF ABS DISC>EPS THEN  GO TO
 7440
7380 IF ABS CA>EPS THEN  GO TO 7
410
7390 IF ABS XF<EPS THEN  GO TO 8
040
7400 GO TO 7500
7410 LET LAMBDA=XF/CA
7420 IF ABS (XF+LAMBDA*CB)<EPS T
HEN  GO TO 8040
7430 GO TO 7500
7440 LET LAMBDA=(CA*YF+CB*XF)/DI
SC
7449 REM Si la linea no corta la
 cara K comprueba la siguiente c
ara
7450 IF LAMBDA<-EPS THEN  GO TO
7460 IF LAMBDA>1+EPS THEN  GO TO
7470 LET MU=(YE*XF-XE*YF)/DISC
7479 REM Si hay interseccion ver
dadera renueva MAX y MIN
7480 IF MAX<MU THEN  LET MAX=MU
7490 IF MIN>MU THEN  LET MIN=MU
7500 LET S1=S2
7510 LET VX1=VX2: LET WY1=WY2
7520 NEXT M
7529 REM Comprueba si la interse
ccion esta entre los extremos de
 la linea
7530 IF MIN>1 THEN  GO TO 8040
7540 IF MAX<0 THEN  GO TO 8040
7550 IF MAX>1 THEN  LET MAX=1
7560 IF MIN<0 THEN  LET MIN=0
7570 IF MAX-MIN<EPS THEN  GO TO
7579 REM Calcula XMID e YMID
7580 LET MID=(MAX+MIN)*0.5: LET
MUD=1-MID
7590 LET XMID=MUD*X1+MID*X2
7600 LET YMID=MUD*Y1+MID*Y2
7610 LET DENOM=PPD*(X(IV2)-X(IV1
))-XMID*(Z(IV2)-Z(IV1))
```

```
7620 IF ABS DENOM<EPS THEN  GO T
O 7650
7629 REM Calcula PHI y por tanto
 XHAT, YHAT, y ZHAT
7630 LET PHI=(XMID*Z(IV1)-PPD*X(
IV1))/DENOM
7640 GO TO 7670
7650 LET DENOM=PPD*(Y(IV2)-Y(IV1
))-YMID*(Z(IV2)-Z(IV1))
7660 LET PHI=(YMID*Z(IV1)-PPD*Y(
IV1))/DENOM
7670 LET ZHAT=(1-PHI)*Z(IV1)+PHI
*Z(IV2)
7680 LET FACT=ZHAT/PPD
7690 LET XHAT=XMID*FACT: LET YHA
T=YMID*FACT
7699 REM Calcula los coeficiente
s del plano que contiene a la ca
ra; A,B,C,D
7700 LET JV1=F(1,K): LET JV2=F(2
,K): LET JV3=F(3,K)
7710 LET DX1=X(JV1)-X(JV2)
7720 LET DX3=X(JV3)-X(JV2)
7730 LET DY1=Y(JV1)-Y(JV2)
7740 LET DY3=Y(JV3)-Y(JV2)
7750 LET DZ1=Z(JV1)-Z(JV2)
7760 LET DZ3=Z(JV3)-Z(JV2)
7770 LET A=DY1*DZ3-DY3*DZ1
7780 LET B=DZ1*DX3-DZ3*DX1
7790 LET C=DX1*DY3-DX3*DY1
7800 LET D=A*X(JV1)+B*Y(JV1)+C*Z
(JV1)
7810 LET S1=A*XHAT+B*YHAT+C*ZHAT
-D
7819 REM Si la cara tapa parte d
e la linea cambia la matriz L
7820 IF ABS S1<EPS THEN  GO TO 8
040
7830 IF ABS (SGN S1+SGN D)<2 THE
N  GO TO 8040
7840 LET MORE=NRL
7850 FOR M=1 TO NRL
7860 LET R1=L(1,M): LET R2=L(2,M
)
7870 IF (R1>MAX) OR (R2<MIN) THE
```

```
N  GO TO 7960
7880 IF (R1>=MIN)  AND (R2<=MAX)
 THEN  GO TO 7950
7890 IF (R1<MIN)  AND (R2>MAX) T
HEN  GO TO 7920
7900 IF (R1<MIN) THEN  GO TO 7940
7910 LET L(1,M)=MAX: GO TO 7960
7920 LET MORE=MORE+1
7930 LET L(1,MORE)=MAX: LET L(2,
MORE)=R2
7940 LET L(2,M)=MIN: GO TO 7960
7950 LET L(1,M)=-1
7960 NEXT M
7969 REM Ordena la matriz L
7970 LET NRL=0
7980 FOR M=1 TO MORE
7990 IF L(1,M)<-EPS THEN  GO TO
8000 LET NRL=NRL+1
8010 LET L(1,NRL)=L(1,M): LET L(
2,NRL)=L(2,M)
8020 NEXT M
8030 IF NRL=0 THEN  GO TO 8160
8040 NEXT K
8049 REM Dibuja las partes visib
les de la linea (si existen)
8050 FOR K=1 TO NRL
8060 LET R1=L(1,K): LET R2=1-R1
8070 LET XP1=X1*R2+X2*R1
8080 LET YP1=Y1*R2+Y2*R1
8090 LET R1=L(2,K): LET R2=1-R1
8100 LET XP2=X1*R2+X2*R1
8110 LET YP2=Y1*R2+Y2*R1
8120 IF (ABS (XP1-XP2)<EPS) AND
(ABS (YP1-YP2)<EPS) THEN  GO TO
8150
8130 LET XPT=XP1: LET YPT=YP1: G
O SUB moveto
8140 LET XPT=XP2: LET YPT=YP2: G
O SUB lineto
8150 NEXT K
8160 LET IV1=IV2: LET X1=X2: LET
 Y1=Y2
8170 NEXT J
8180 NEXT I
8190 RETURN
```

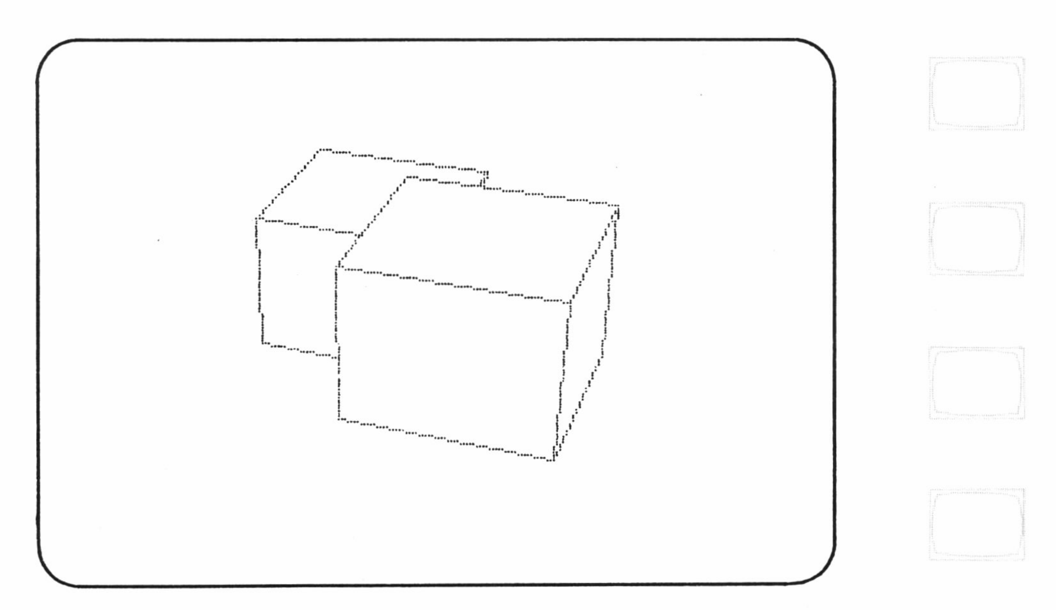


Figura 12.1

Ejemplo 12.1

Dibujaremos una versión en perspectiva con líneas ocultas de la escena que en principio se dibujó en la figura 9.2: es decir, uno de los dos cubos mostrados en la figura 12.1. La escena tiene los valores HORIZ = 9, VERT = 6 y se observa desde (15, 10, 5) hasta (0, 0, 0).

Utilizaremos "rut1", "rut3" y el algoritmo general de líneas ocultas (*hidden,* listado 12.1) junto con la subrutina "escena3", y "cubo" dadas en el listado 12.2. Con esta última versión de "cubo" hemos considerado todos los métodos matriciales de construcción de un objeto; almacenado/no almacenado, líneas/caras. Estamos utilizando intencionadamente la figura del cubo una y otra vez en nuestros diagramas, ya que se trata de un objeto muy sencillo que resulta fácil de visualizar en sus distintas construcciones, y por tanto no complica nuestra discusión acerca de los principios generales de gráficos tridimensionales. Ahora es el momento de introducir mayor complejidad en nuestros objetos: suponiendo que se han comprendido las limitaciones de los algoritmos, las ideas expuestas son igualmente válidas para figuras más complejas. Aquellos que posean un Spectrum 16K, encontrarán que, por desgracia, los programas de gran complejidad no pueden acomodarse en sus ordenadores. Deberán partir dichos programas en secciones independientes y almacenar los datos de los objetos y las matrices temporalmente en cinta.

Listado 12.2

```
6000 REM escena3/figura 12.1
6010 DIM X(16): DIM Y(16): DIM Z
(16)
```

```
6020 DIM V(16): DIM W(16): DIM F
(4,12): DIM H(12)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4): DIM Q(4,4)
6040 LET cube=6500: LET hidden=7
000
6050 LET PPD=3*VERT: LET NOV=0:
LET NOF=0
6059 REM Situa el primer cubo en
 la posicion observada
6060 GO SUB idR3: GO SUB look3
6070 GO SUB cube
6079 REM Copia la matriz de paso
 de posicion actual a observada
en Q
6080 FOR I=1 TO 4: FOR J=1 TO 4
6090 LET Q(I,J)=R(I,J)
6100 NEXT J: NEXT I: GO SUB idR3
6109 REM Calcula la matriz de pa
so de inicial a actual
6110 LET TX=3: LET TY=1.5: LET T
Z=2: GO SUB tran3: GO SUB mult3
6119 REM Restablece la matriz de
 actual a observada
6120 FOR I=1 TO 4: FOR J=1 TO 4
6130 LET A(I,J)=Q(I,J)
6140 NEXT J: NEXT I
6149 REM Calcula la matriz de in
icial a observada
6150 GO SUB mult3
6159 REM Situa el segundo cubo y
 dibuja la vista de la escena co
n lineas ocultas
6160 GO SUB cube
6170 GO SUB hidden
6180 RETURN

6500 REM cubo/vertices y caras (
almacenadas)
6501 REM Datos de entrada PPD,NO
V,NOF,X(NOV),Y(NOV),Z(NOV),V(NOV
),W(NOV)
6502 REM F(4,NOF),H(NOF),R(4,4)
6503 REM Datos de salida NOV,NOF
,X(NOV),Y(NOV),Z(NOV),V(NOV),W(N
```

```
OV)
6504 REM F(4,NOF),H(NOF)
6510 DATA 1,1,1, 1,1,-1, 1,-1,-1
, 1,-1,1, -1,1,1, -1,1,-1, -1,-1
,-1, -1,-1,1
6520 DATA 1,2,3,4, 5,8,7,6, 1,5,
6,2, 2,6,7,3, 3,7,8,4, 4,8,5,1
6530 RESTORE cube
6539 REM Extiende la base de dat
os de los vertices en la posicio
n observada.
6540 LET NV=NOV
6550 FOR I=1 TO 8
6560 READ XX,YY,ZZ: LET NOV=NOV+1
6570 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6580 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6590 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6599 REM Transforma la perspecti
va
6600 LET V(NOV)=PPD*X(NOV)/Z(NOV)
6610 LET W(NOV)=PPD*Y(NOV)/Z(NOV)
6620 NEXT I
6629 REM Lee y extiende la base
de datos de las caras
6630 FOR I=1 TO 6
6640 READ F1,F2,F3,F4: LET NOF=N
OF+1
6650 LET H(NOF)=4
6660 LET F(1,NOF)=F1+NV: LET F(2
,NOF)=F2+NV
6670 LET F(3,NOF)=F3+NV: LET F(4
,NOF)=F4+NV
6680 NEXT I
6690 RETURN
```

Ejercicio 12.1

Constrúyanse escenas con líneas ocultas compuestas de cubos, tetraedros, pirámides, octaedros, cuboctraedros, icosaedros. Como ayuda, facilitamos los listados 12.3 y 12.4, que son subrutinas de construcción de un cuboctaedro y un icosaedro. Es-

criba sus propias subrutinas para un octaedro, o, quizá, para objetos más complicados como un rombododecaedro.

Listado 12.3

```
6500 REM icos/aedro
6501 REM Datos de entrada y sali
da como en el cubo anterior
6510 DATA 0,1,T, T,0,1, 1,T,0, 0
,-1,T, T,0,-1, -1,T,0, 0,1,-T, -
T,0,1, 1,-T,0, 0,-1,-T, -T,0,-1,
 -1,-T,0
6520 DATA 1,3,2, 1,2,4, 1,4,8, 1
,8,6, 1,6,3, 2,3,5, 2,9,4, 4,12,
8, 8,11,6, 3,6,7, 2,5,9, 4,9,12,
 8,12,11, 6,11,7, 3,7,5, 5,10,9,
 9,10,12, 12,10,11, 11,10,7, 7,1
0,5
6530 RESTORE icosa: LET T=(1+SQR
 5)/2
6540 LET NV=NOV
6550 FOR I=1 TO 12
6560 READ XX,YY,ZZ: LET NOV=NOV+
1
6570 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6580 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6590 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6600 LET V(NOV)=X(NOV)*PPD/Z(NOV
)
6610 LET W(NOV)=Y(NOV)*PPD/Z(NOV
)
6620 NEXT I
6630 FOR I=1 TO 20
6640 READ F1,F2,F3: LET NOF=NOF+
1
6650 LET H(NOF)=3
6660 LET F(1,NOF)=F1+NV: LET F(2
,NOF)=F2+NV: LET F(3,NOF)=F3+NV
6670 NEXT I
6680 RETURN
```

Listado 12.4

```
6700 REM cuboct/aedro
6701 REM Datos de entrada y sali
da como en el cubo anterior
6710 DATA 0,1,1, 1,0,1, 1,1,0, 0
,-1,1, 1,0,-1, -1,1,0, 0,1,-1, -
1,0,1, 1,-1,0, 0,-1,-1, -1,0,-1,
 -1,-1,0
6720 DATA 1,2,4,8, 1,6,7,3, 2,3,
5,9, 4,9,10,12, 5,7,11,10, 6,8,1
2,11
6730 DATA 1,3,2, 1,8,6, 2,9,4, 3
,7,5, 4,12,8, 5,10,9, 6,11,7, 10
,11,12
6740 RESTORE cuboct
6750 LET NV=NOV
6760 FOR I=1 TO 12
6770 READ XX,YY,ZZ: LET NOV=NOV+
1
6780 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6790 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6800 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6810 LET V(NOV)=X(NOV)*PPD/Z(NOV
)
6820 LET W(NOV)=Y(NOV)*PPD/Z(NOV
)
6830 NEXT I
6840 FOR I=1 TO 6
6850 READ F1,F2,F3,F4: LET NOF=N
OF+1
6860 LET H(NOF)=4
6870 LET F(1,NOF)=F1+NV: LET F(2
,NOF)=F2+NV: LET F(3,NOF)=F3+NV:
 LET F(4,NOF)=F4+NV
6880 NEXT I
6890 FOR I=1 TO 8
6900 READ F1,F2,F3: LET NOF=NOF+
1
6910 LET H(NOF)=3
6920 LET F(1,NOF)=F1+NV: LET F(2
```

```
,NOF)=F2+NV: LET F(3,NOF)=F3+NV
6930 NEXT I
6940 RETURN
```

Ejemplo **12.2**

Al llegar aquí habrá observado que los algoritmos generales de líneas ocultas son programas muy lentos: necesitan realizar un enorme número de comparaciones. El dibujo de los dos cubos de la figura 12.1 lleva al menos cinco minutos. Esto significa que estaremos bastante limitados en el tipo de objetos que podremos realizar. Sin embargo, resulta un excelente ejercicio, y si usted tiene oportunidad de utilizar ordenadores mayores, observará que los algoritmos anteriores funcionan también en ellas, aunque mucho más rápido. Damos en el listado 12.5 una subrutina "escena3" y en los listados 12.6 y 12.7, ejemplos de dos objetos tridimensionales en forma de estrella (en ambos se requiere un parámetro A que varía la elongación de las puntas). Estas dos subrutinas "estrella' se basan en tetraedros y cubos. La figura 12.2 ha sido dibujada con HORIZ = 48, VERT = 32, vista desde (35, 20, 25) hacia (0, 0, 0).

Listado 12.5

```
6000 REM escena3/dos estrellas s
in lineas ocultas
6010 DIM X(22): DIM Y(22): DIM Z
(22)
6020 DIM V(22): DIM W(22): DIM F
(3,36): DIM H(36)
6030 DIM A(4,4): DIM B(4,4): DIM
 R(4,4): DIM Q(4,4)
6040 LET star1=6500: LET star2=6
700: LET hidden=7000
6050 LET PPD=3*VERT: LET NOV=0:
LET NOF=0
6059 REM Situa la primera estrel
la
6060 GO SUB idR3: GO SUB look3
6070 LET A=6: GO SUB star1
6080 FOR I=1 TO 4: FOR J=1 TO 4
6090 LET Q(I,J)=R(I,J)
6100 NEXT J: NEXT I: GO SUB idR3
6109 REM Situa la segunda estrel
la
6110 LET TX=5: LET TY=5: LET TZ=
```

```
5: GO SUB tran3: GO SUB mult3
6120 FOR I=1 TO 4: FOR J=1 TO 4
6130 LET A(I,J)=Q(I,J)
6140 NEXT J: NEXT I
6150 GO SUB mult3
6160 LET A=4: GO SUB star2
6170 GO SUB hidden
6180 RETURN
```

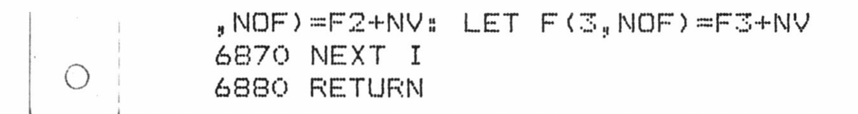
Listado 12.6

```
6500 REM estrella1
6501 REM Datos de entrada y sali
da como en el cubo anterior
6509 REM Estrella basada en un c
ubo
6510 DATA 1,1,1, 1,1,-1, 1,-1,-1
, 1,-1,1, -1,1,1, -1,1,-1, -1,-1
,-1, -1,-1,1, A,0,0, -A,0,0, 0,A
,0, 0,-A,0, 0,0,A, 0,0,-A
6520 DATA 1,2,9, 2,3,9, 3,4,9, 4
,1,9, 6,5,10, 5,8,10, 8,7,10, 7,
6,10, 2,1,11, 1,5,11, 5,6,11, 6,
2,11, 4,3,12, 3,7,12, 7,8,12, 8,
4,12, 1,4,13, 4,8,13, 8,5,13, 5,
1,13, 3,2,14, 2,6,14, 6,7,14, 7,
3,14
6530 RESTORE star1
6540 LET NV=NOV
6550 FOR I=1 TO 14
6560 READ XX,YY,ZZ: LET NOV=NOV+
1
6570 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6580 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6590 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6600 LET V(NOV)=X(NOV)*PPD/Z(NOV
)
6610 LET W(NOV)=Y(NOV)*PPD/Z(NOV
)
6620 NEXT I
```

```
6630 FOR I=1 TO 24
6640 READ F1,F2,F3: LET NOF=NOF+
1
6650 LET H(NOF)=3
6660 LET F(1,NOF)=F1+NV: LET F(2
,NOF)=F2+NV: LET F(3,NOF)=F3+NV
6670 NEXT I
6680 RETURN
```

Listado 12.7

```
6700 REM estrella2
6701 REM Datos de entrada y sali
da como en el cubo anterior
6709 REM Estrella basada en un t
etraedro
6710 DATA 1,1,1, 1,-1,-1, -1,1,-
1, -1,-1,1, -A,-A,-A, -A,A,A, A,
-A,A, A,A,-A
6720 DATA 2,1,8, 3,2,8, 1,3,8, 1
,2,7, 4,1,7, 2,4,7, 2,3,5, 4,2,5
, 3,4,5, 3,1,6, 4,3,6, 1,4,6
6730 RESTORE star2
6740 LET NV=NOV
6750 FOR I=1 TO 8
6760 READ XX,YY,ZZ: LET NOV=NOV+
1
6770 LET X(NOV)=XX*R(1,1)+YY*R(1
,2)+ZZ*R(1,3)+R(1,4)
6780 LET Y(NOV)=XX*R(2,1)+YY*R(2
,2)+ZZ*R(2,3)+R(2,4)
6790 LET Z(NOV)=XX*R(3,1)+YY*R(3
,2)+ZZ*R(3,3)+R(3,4)
6800 LET V(NOV)=X(NOV)*PPD/Z(NOV
)
6810 LET W(NOV)=Y(NOV)*PPD/Z(NOV
)
6820 NEXT I
6830 FOR I=1 TO 12
6840 READ F1,F2,F3: LET NOF=NOF+
1
6850 LET H(NOF)=3
6860 LET F(1,NOF)=F1+NV: LET F(2
```



```
,NOF)=F2+NV: LET F(3,NOF)=F3+NV
6870 NEXT I
6880 RETURN
```

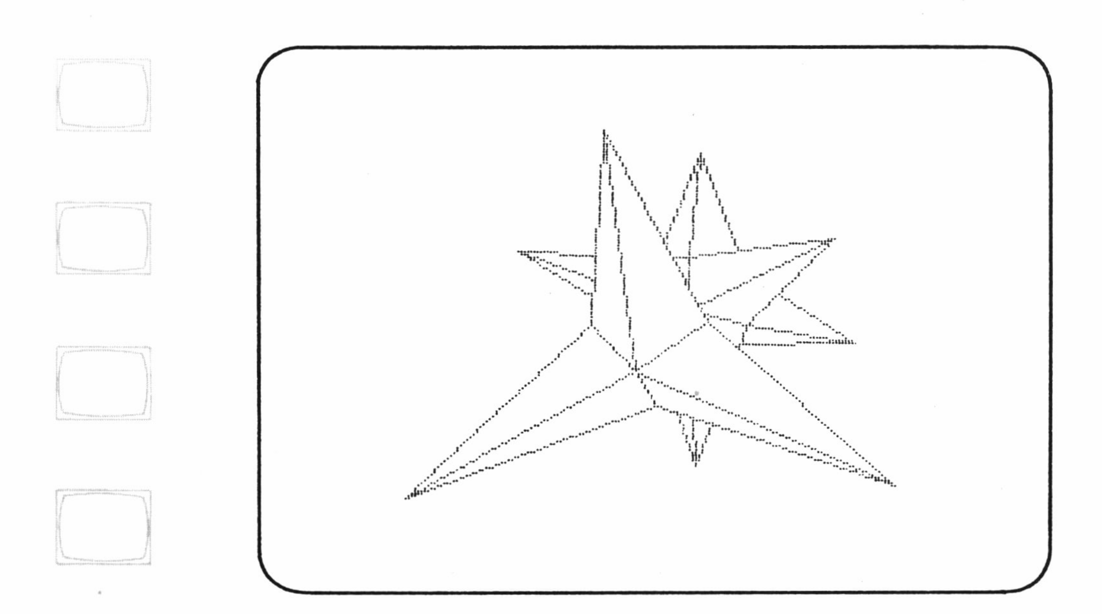


Figura 12.2

Ejercicio 12.2

El programa del listado 10.1 comprueba que el orden de los vértices de una cara triangular es antihorario. El programa está diseñado para ser utilizado con cuerpos convexos que contengan el origen. Extienda este programa de manera que pueda abarcar el caso más general: es decir, especificando la posición del observador y las coordenadas de un punto dentro del objeto (no necesariamente el origen) de manera que dicho punto y el observador se sitúen en lados opuestos de un plano infinito que contenga la cara. Utilice este programa para comprobar los objetos en forma de estrella que acabamos de ver (de hecho, para estas figuras el origen puede utilizarse como punto interior).

Produzca a continuación sus propios objetos en forma de estrella, basados en octaedros, cuboctaedros, icosaedros o dodecaedros. También se pueden producir estrellas basadas en cuerpos de revolución muy sencillos, y además no tenemos por qué limitarnos a objetos simétricos. Si se trata de formas no simétricas necesitará realmente la versión extendida del listado 10.1. Si se mantiene dentro de los límites marcados por las restricciones mencionadas con anterioridad, el listado 12.1 podrá dibujar cualquier forma.

Ejercicio 12.3

Añada información extra a los objetos que está colocando. Además de los datos de vértices y caras, introduzca otra matriz L de tal forma que L(1, I) y L(2, I) contengan los índices de la dos caras cuya intersección es la arista I-ésima, 1 ⩽ I ⩽ NOL. Altere a continuación el algoritmo de líneas ocultas de manera que ignore cualquier línea limitada por dos caras invisibles (horarias), y no compare una cara con las líneas que se sitúan en la misma.

Si usted ha leído y comprendido los capítulos 7 al 12, encontrará que hemos alcanzado los límites de gráficos tridimensionales en el Spectrum. Deberá tener acceso a ordenadores mayores si desea avanzar en el estudio de este tipo de gráficos por ordenador. Además, deberá estudiar las técnicas que utilizan estructuras de datos, en vez de matrices, para definir las escenas. Por ejemplo, se puede definir una escena completa como una lista encadenada de apuntadores *(pointers),* cada uno de ellos apuntando a una lista de información sobre las caras de un tipo particular de objetos. Las propias caras pueden almacenarse como listas de vértices: es una idea aparentemente compleja, pero hace que los algoritmos de líneas ocultas resulten mucho más sencillos. En las propias listas están almacenadas implícitamente las relaciones entre objetos y caras. Cuando avance en ese estudio, podrá aventurarse con complicados algoritmos gráficos que incluyan métodos de animación, coloreado y sombreado. Recomendamos que se acceda a algún libro especializado en la materia, tanto para el estudio de estructuras de datos como para métodos gráficos complejos. En el siguiente capítulo realizaremos un repaso de gráficos de caracteres más avanzados, e introduciremos un método de producción de dibujos animados tridimensionales.

Programas completos

I. "rut1", "rut3", los listados 12.1, "algoritmo general de líneas ocultas" *(idden)* y 12.2 "escena3" y "cubo" *(scene3* y Datos necesarios: HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ). Intente *a)* 9, 6, (15, 10, 5), (0, 0, 0); *b)* 9, 6, (– 10, 10, – 10), (0, 1, 0).

II. "rut1", "rut3", los listados 12.1, "algoritmo general de líneas ocultas" *(hidden)* y 12.2, "escena3" *(scene3).* Introduzca por medio de un MERGE el listado 12.3, "icosaedro" *(icosa)* y 12.4, "cuboctaedro" *(cuboct)* y altere la subrutina "escena3" como sigue:

```
6010 DIM X(24): DIM Y(24): DIM Z
(24)
6020 DIM V(24): DIM W(24): DIM F
(4,34): DIM H(34)

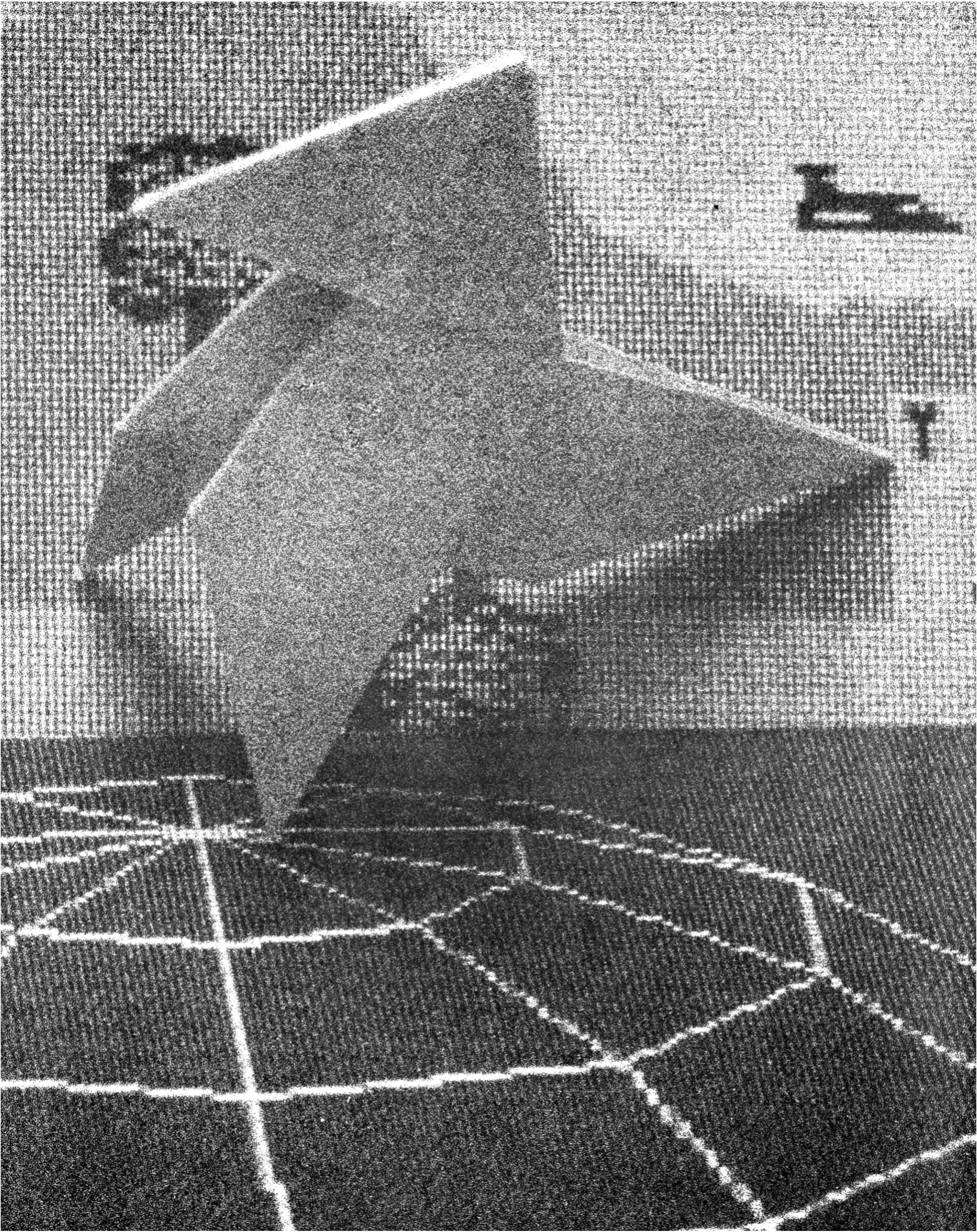
6040 LET cuboct=6700: LET icosa=
6500: LET hidden=7000
```

```
6070 GO SUB icosa

6160 GO SUB cuboct
```

Datos necesarios: HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ). Intente *a*) 9, 6, (15, 10, 5), (0, 0, 0); *b*) 9, 6, (– 10, 10, 10), (0, 1, 0).

III. "rut1", "rut3", los listados 12.1, 12.5 "escena3" *(scene3),* 12.6, "estrella1" *(star1)* y 12.7 "estrella2" *(star2)*. Datos requeridos: HORIZ, VERT, (EX, EY, EZ), (DX, DY, DZ). Intente *a*) 60, 40, (10, 20, 30), (0, 0, 0); *b*) 90, 60, (– 10, 20, – 10), (0, 1, 0).



13
Técnicas avanzadas de programación

Desde hace algunos años, se ha popularizado el término “amistoso al usuario” *(user friendly)* para denotar programas sencillos de utilizar incluso para personal no especializado. Lo que realmente da a un programa calidad profesional es precisamente el hecho de ser *user friendly*. Quizá es una de los pocos *slogans* de la jerga publicitaria que tiene alguna relación con la realidad: es necesario hacer programas que sean fáciles de utilizar, no sólo para nosotros sino para otros usuarios. Todos hemos pasado la desagradable experiencia de intentar utilizar un programa que escribimos rápidamente tres meses antes, para encontrarnos con que está tan mal estructurado y/o comentado que somos incapaces de comprenderlo nosotros mismos. Es una buena práctica de programación utilizar comentarios así como hacer entradas/salidas autoexplicativas. Asegúrese de que los mensajes que aparecen en pantalla en la ejecución son claros y concisos. Otra forma de hacer más accesible un programa es introducir una pequeña subrutina explicativa al comienzo de la ejecución del estilo de las utilizadas en los videojuegos comerciales.

Cuando tengamos un programa que dispone de un conjunto de subrutinas que se pueden utilizar en cualquier secuencia o combinación, el método usual de selección es el *menú* (véase el programa GENERADOR DE CARACTERES del capítulo 5). Si conseguimos que los mensajes sean apropiados y descriptivos de las acciones a realizar, este método es especialmente útil para usuarios que no comprenden los detalles del programa y desean utilizarlo únicamente como herramienta de dibujo. En la mayor parte de las veces, el tipo de mensaje a mostrar es un asunto puramente de sentido común. Evite los clásicos mensajes confusos como PULSE 1 PARA DATOS DUPLICADOS, 2 PARA DATOS SIMPLES. Si es posible, utilice las teclas de cursor para movimientos en la pantalla (véase opción 3 del programa GENERADOR DE CARACTERES del capítulo 5): esta técnica resultará natural a cualquier usuario

normal de Spectrum. El listado 13.1 muestra un programa que puede utilizarse para dibujar polígonos (como en el ejercicio 1.3). La subrutina de entrada utiliza "EDIT" y las teclas de cursor, estando diseñados para personas familiarizadas con la programación en Spectrum.

Listado 13.1

```
  10 DIM X(10): DIM Y(10)
  20 LET input=200: LET list=300
  30 GO SUB input
  40 PLOT X(10),Y(10): LET OX=X(
10): LET OY=Y(10)
  50 FOR I=1 TO 10: DRAW X(I)-OX
,Y(I)-OY: LET OX=X(I): LET OY=Y(
I): NEXT I
  60 PAUSE 250: GO TO 30

 200 REM entrada de coordenadas
 210 LET I=1
 220 GO SUB list
 230 LET I$=INKEY$: IF I$="" THE
N  GO TO 230
 240 IF I$=CHR$ 10 AND I<11 THEN
  LET I=I+1: GO TO 220
 250 IF I$=CHR$ 11 AND I>1 THEN
 LET I=I-1: GO TO 220
 260 IF I$<>CHR$ 7 THEN  GO TO 2
30
 270 IF I=11 THEN  INPUT "Fin. ?
 "; LINE I$: IF I$="s" THEN  CLS
 : RETURN
 280 IF I=11 THEN  GO TO 220
 290 INPUT "Coord. X";X(I),"Coor
d. Y";Y(I): GO TO 220

 300 REM lista los datos
 310 CLS : FOR J=1 TO 10
 320 PRINT AT J,1;"Coord. X";X(J
),"Coord. Y";Y(J)
 330: NEXT J: PRINT AT 11,1;"Fin
.": PRINT AT I,0;">"
 340 RETURN
```

Códigos de control

El comando INPUT se puede utilizar para obtener mensajes precediendo a cualquier petición de datos. En la mayor parte de los casos, simplemente colocamos el mensaje encerrado entre comillas, de la misma forma en que deseamos que aparezca, justo antes del dato en la sentencia INPUT. En el Spectrum podemos forzar la evaluación de cualquier expresión de tipo tira de caracteres *(string),* incluyendo llamadas a funciones, colocando estas expresiones entre paréntesis. De esta forma serán tratadas como si estuviesen encerradas entre comillas. Se ha usado este método en el listado anterior; podemos aprovecharlo para producir mensajes mucho más complicados. Si construimos una tira de caracteres que contenga códigos de control podremos representar gráficos coloreados en cualquier lugar de la pantalla. Además, una variable de este tipo se puede construir en varias etapas y colocar el nombre a continuación en el comando INPUT entre paréntesis. Se pueden incluir códigos de control en la tira usando la función CHR$, y variables numéricas usando STR$. Esta técnica se utiliza en el programa MASTER MIND (listado 5.6) y también en el JUEGO DEL GUSANO (listado 1.16). Se pueden conseguir cambios rápidos en cualquier parte de la pantalla utilizando tiras con códigos de color incluidos o instrucciones del tipo PRINT AT CODE. En el listado 13.2, como ejemplo, presentamos un programa menú de un eventual sistema de manejo de ficheros. Obsérvese la utilización de colores para advertir de opciones potencialmente peligrosas.

Se pueden introducir también los códigos de control directamente desde teclado para su inclusión en tiras (véase Manual del Spectrum). Estos códigos directos son ignorados durante la ejecución; pueden también formar parte de sentencias de programa con el fin de destacar u ocultar partes del listado. En los programas de la cinta que acompaña al libro se encuentran varios ejemplos en los cuales las sentencias REM de comienzo de cada subrutina empiezan con un CODE para BRIGHT 1 ("9" en modo extendido) y terminan con BRIGHT 0 ("8" en modo extendido). Para rellenar la sentencia REM de modo que finalice en el borde de la pantalla, necesitamos introducir un CODE 6 en la línea, que equivale a una coma en sentencias PRINT. Aunque esta opción no parece en principio accesible desde teclado, podemos conseguir el efecto deseado insertando el CODE de PAPER 6 ("6" en modo extendido) y pulsando inmediatamente después DELETE. Esta secuencia elimina el CODE de PAPER pero deja CODE 6, que tiene el efecto subsiguiente de hacer que el cursor salte a la siguiente mitad o línea completa.

Así, CODE 6 se interpreta tal como es, y no como parámetro para PAPER. Si utilizados CODE 6 al final de cada línea antecediendo una nueva sección del programa obligaremos a que en la salida en pantalla aparezca una línea en blanco en el listado. Por otra parte, podemos ocultar la palabra REM colocando un CODE para INK 0 ("0" en modo extendido) detrás de REM, y otro INK 7 ("7" en modo extendido) delante del mismo.

Como norma general, es mejor escribir los programas en módulos. Dentro de los mismos conviene destacar los nombres de los módulos, lo cual nos permitirá leer, corregir y adaptar los programas con facilidad. Para demostrar cómo se mejora la legibilidad de un listado, obsérvese la figura 13.1, que muestra parte del listado 13.6 tal como aparece en la pantalla.

Listado 13.2

```
 100 DIM G(6): DIM A$(6,24)
 110 FOR I= 1 TO 6: READ G(I): N
EXT I
 120 DATA 1000,1000,1000,1000,10
00,1000
 130 LET A$(1) ="EDITAR": LET A$
(2)= "IMPRIMIR"
 140 LET A$(3)= "GUARDAR": LET A
$(4)="CARGAR"
 150 LET A$(5) = CHR$ 17 + CHR$
16 + CHR$ 7 + "BORRAR FICHERO" +
 CHR$ 17 + CHR$ 7
 160 LET A$(6) = CHR$ 17 + CHR$
2 + CHR$ 16 + CHR$ 7 + "BORRAR T
ODOS" + CHR$ 17 + CHR$ 7

 500 REM menu/seleccion de opcio
nes
 510 CLS : PRINT AT 2,8;"MANEJO
DE FICHEROS"
 520 FOR I = 1  TO 6
 530 PRINT AT I*2 + 4,8;I;"    ";
A$(I)
 540 NEXT I
 550 INPUT "ELIJA OPCION";OP
 560 IF OP < 1  OR OP > 6 THEN
GO TO 550
 570 IF OP < 5 THEN  GO TO 600
 580 INPUT ("ESTA SEGURO DE"; +
CHR$ 6 + A$(OP) + " ");Y$
 590  IF Y$ <> "s" THEN  GO TO 5
50
 600 GO SUB G(OP): GO TO 500

1000 REM resto de subrutinas
1010 RETURN
```

```
   10 REM   cargador para rutina
en codigo maquina
   20 CLEAR 31999
   30 FOR I=0 TO 74
   40 READ A: POKE 32000+I,A
   50 NEXT I

  100 REM desplazamiento de panta
lla hacia abajo (zona graficos)
  110 DATA 1,175,0
 :  REM LD BC,175
  120 DATA 221,33,75,125
 :  REM LD IX,LOBYTE
  130 DATA 221,9
 :  REM ADD IX,BC
  140 DATA 221,110,0
 :  REM LD L,(IX+0)
  150 DATA 221,33,251,125
 :  REM LD IX,HIBYTE
  160 DATA 221,9
 :  REM ADD IX,BC
  170 DATA 221,102,0
```

Figura 13.1

Estructura del display file

Sabemos que el *display file* se puede alterar directamente desde programas BASIC, así como examinar un contenido; sin embargo, para hacer uso de esta facilidad, debemos comprender el modo en que el *display file* está organizado. Cada línea horizontal de la pantalla se compone de 32 bytes, cada uno de ellos de ocho bits. Estos 32 bytes están almacenados en posiciones consecutivas de la memoria. Sin embargo, los bytes de la línea siguiente se almacenan en la posición equivalente de la siguiente página de la memoria. Para un microprocesador de ocho bits, como lo es el Z80 que contiene el ZX Spectrum, una *página* de memoria significa 2^8 ó 256 bytes: utilizamos el identificador PAGE para referirnos a esta cantidad. Se utiliza esta disposición para ayudar a la circuitería de video a trabajar eficientemente con la pantalla. Así pues, cada página tiene espacio suficiente para ocho líneas de 32 bytes. La primera página del *display file* contiene la línea superior de cada una de las ocho primeras líneas de bloques de caracteres. Las restantes siete líneas de estas filas residen en otras siete páginas de memoria, conteniendo cada página los datos de la línea correspondiente. Después de estas ocho páginas existen otros dos conjuntos de ocho páginas cada uno que contienen la información del resto de la pantalla. A partir de estos datos podemos calcular los bytes que contienen los datos de la línea superior de un bloque determinado, y saber que cada una de las demás líneas se almacenan 256 bits (una página) más adelante. Para averiguar la posición de la línea superior de un bloque determinado en el *display file,* deberemos saber también a qué tercio de la pantalla (ocho páginas) pertenece el bloque. Necesitamos también identificar, dentro de este tercio de pantalla, a cuál de los 256 bloques

posibles nos estamos refiriendo. La siguiente función calcula la posición de la primera línea de un bloque de caracteres, dado la fila R y columna C:

DEF FN A(R, C) = 16384 + INT (R/8)*2048 + (R − INT(R/8)*8)*32 + C

Esta función se compone de cuatro partes:

16384 (el comienzo del *display file*)

+ INT (R/8)*2048 (para situarnos en el tercio de pantalla (0, 1, 2) correspondiente multiplicando por la longitud de un tercio de la pantalla (2048 = = 8*PAGE))

+ (R − INT(R/8)*8)*32 (más la posición de la fila dentro de ese tercio de pantalla (0-7) multiplicada por 32 (columnas por filas))

+ C (más la columna (0-31) dentro de dicha fila)

Las dos primeras partes determinan la localización del comienzo de página que contiene las primeras líneas de ese tercio de pantalla. Los dos últimos localizan el byte correspondiente entre los 256 de la página.

Utilizando una nueva función con un carácter como parámetro de entrada, podemos encontrar el comienzo del dato que define a dicho carácter, y transferir estos datos a posiciones del *display file*

DEF FN T(A$) = PEEK 23606 + PEEK 23607*256 + 8*CODE A$

Esta función utiliza la variable de sistemas CHARS (almacenada en las posiciones 23606 y 23607, véase capítulo 5) para localizar el comienzo de la tabla de datos del conjunto de caracteres. A continuación añade CODE multiplicado por ocho para el carácter requerido y así encuentra la dirección de la primera parte del dato. Obsérvese que estas dos funciones pueden utilizarse para escribir en las dos líneas inferiores de la pantalla que no son normalmente accesibles; demostramos este último punto en el programa del listado 13.3.

Listado 13.3

```
 100 REM programa principal
 110 LET print = 500
 120 LET p$ = "9 sentencia FALSA
, 120:1": LET ROW = 23: LET COL
= 0
 130 GO SUB print
 140 LET P$= "imprime en cualqui
```

```
er parte": LET ROW = 7: LET COL
= 16
 150 GO SUB print
 160 PAUSE 250: STOP

 500 REM subrutina de impresion
 510 DEF FN A(R,C) = 16384 + INT
 (R/8)*2048 + (R - INT (R/8)*8)*
32 + C
 520 DEF FN D(A$) = PEEK 23606 +
 256 *PEEK 23607 + 8*CODE A$
 530 FOR I = 1 TO LEN P$
 540 LET ADRESS =FN A(ROW,COL)
 550 LET DATA1 = FN D(P$(I))
 560 FOR J = 0 TO 7
 570 POKE ADRESS + J*256,PEEK (D
ATA1 + J)
 580 NEXT J
 590 LET COL = COL + 1: IF COL =
 32 THEN  LET COL = 0: LET ROW =
 ROW + 1: IF ROW = 24 THEN  LET
ROW = 0
 600 NEXT I
 610 RETURN
```

Transferencia rápida de datos de pantalla

Si intentamos mover el *display file* del Spectrum rápidamente nos daremos cuenta en seguida de que el BASIC no tiene la velocidad suficiente para transferir la cantidad requerida de datos adecuadamente. Sin embargo, el Spectrum está controlado por un microprocesador Z80 que se halla especialmente dotado para la tarea de mover datos de un lado a otro con rapidez. Para programar directamente estos movimientos de datos en lugar de utilizar BASIC como intermediario) necesitamos hacer una pequeña excursión al *código máquina.* El código máquina Z80 tiene instrucciones equivalentes a PEEK y POKE de BASIC, que se pueden utilizar para examinar o sustituir números en una determinada posición de memoria.

El Z80 tiene también registros internos, en los cuales se colocan los números, y se utilizan de una forma muy semejante a las variables en BASIC.

Podemos, por tanto, escribir un conjunto de instrucciones en código máquina (es decir, un programa) para cargar uno de estos registros con un número de una posición de memoria determinada y a continuación cargar el número del registro en otra posición de memoria. Este proceso puede parecer poco interesante, hasta que uno se percata de que el microprocesador Z80 es capaz de realizar esta operación

un millón de veces por segundo. Desgraciadamente, el código máquina resulta un extraño jeroglífico al principiante; tiene la apariencia de un chorro de números aparentemente sin conexión entre ellos. Resulta más conveniente usar *lenguaje ensamblador,* que contiene palabras cortas, *códigos mnemónicos,* que resultan de gran ayuda para comprender la función de cada instrucción de código máquina (sabiendo un poco de inglés). La estructura del Spectrum requiere que las instrucciones de código máquina se almacenen en sentencias DATA y a continuación se inserten en memoria por medio de POKE. Resulta útil incluir las instrucciones de lenguaje ensamblador como sentencias REM en la línea DATA. Se llama *ensamblador* a un programa capaz de traducir lenguaje ensamblador a código máquina; si no disponemos de uno de ellos, podemos utilizar la tabla que se da en el apéndice A del Manual del Spectrum. Una de las instrucciones más potentes en el microprocesador Z80 es 237, 176, que se simboliza en lenguaje ensamblador con el código **LDIR.** Este código es un *mnemónico* de la expresión inglesa ***LoaD*** *and* ***I****ncrement* ***R****epeated* (cargar e incrementar repetidamente), instrucción utilizada para transferir bloques de datos de un lugar a otro de la memoria. Antes de ejecutar la instrucción, sin embargo, debemos cargar valores apropiados en algunos de los registros. En concreto, se debe introducir en

DE la dirección de destino de los datos.
HL la dirección actual de los datos.
BC el número de bytes a transferir.

Para mayor claridad presentamos en la figura 13.2 un ejemplo de subrutina de código máquina, con su equivalente en lenguaje ensamblador, y además un programa BASIC que ejecuta exactamente la misma función. Recuérdese que el lenguaje ensamblador es simplemente una forma de hacer el código máquina más comprensible. El programa completo, listado 13.4, introduce en la memoria el código máquina por medio de POKE y a continuación llama a la subrutina. Este ejemplo copia el tercio inferior del *display file* en el tercio superior de la pantalla.

```
                                    300 REM transferencia de datos en BASIC
17,0,64   LD     DE,16384           310 LET DE = 16384
33,0,80   LD     HL,20480           320 LET HL = 20480
1,0,8     LD     BC,2048            330 LET BC = 2048
237,176   LDIR                      340 LET A = PEEK HL:POKE DE,A
                                    350 LET DE = DE + 1:LET HL = HL + 1
                                    360 LET BC = BC - 1
                                    370 IF BC <> 0 THEN GO TO 340
201       RET                       380 RETURN
```

Figura 13.2

Una vez introducidas las instrucciones de código máquina en la memoria, ejecutamos la subrutina utilizando la función USR con la dirección de comienzo de aquélla. USR acompañada de una dirección numérica simplemente llama a la subrutina en

código máquina que comienza en tal dirección. La llamada se ejecuta en un programa BASIC por medio de un comando como LET A = USR 32000 (32000 es la dirección que contiene 17, el primer byte del código).

Listado 13.4

```
  10 REM cargador para subrutina
 en codigo maquina
  19 REM para maquinas de 16K us
ar CLEAR 31999
  20 CLEAR 63999
  30 FOR I = 0 TO 11: READ A
  39 REM para 16K utilizar POKE
32000
  40 POKE 64000 + I,A: NEXT I

 100 REM datos para la subrutina
 de transferencia en codigo maqu
ina
 110 DATA 17,0,64             : RE
M  LD      DE,16384
 120 DATA 33,0,80             : RE
M  LD      HL,20480
 130 DATA 1,0,8               : RE
M  LD      BC,2048
 140 DATA 237,176             : RE
M  LDIR
 150 DATA 201                 : RE
M  RET

 200 REM programa principal
 210 CLS : FOR I = 0 TO 21
 220 FOR J =0 TO 31: PRINT AT I,
J;CHR$ (I + 64): NEXT J
 230 NEXT I
 240 PRINT AT 19,1; INVERSE 1;"P
ULSE CUALQUIER TECLA PARA COMENZ
AR"
 250 IF INKEY$="" THEN  GO TO 25
0
 259 REM para 16K cambie a USR 3
2000
 260 LET A = USR 64000: STOP
```

Ejercicio 13.1

Teclee la subrutina BASIC de la figura 13.2 y cronometre el tiempo que tarda en ejecutarse. Compare dicho tiempo con el que transcurre en la ejecución de la subrutina código máquina del listado 13.4. Observará que el código máquina es miles de veces más rápido.

Animación (sólo en ordenadores de 48K)

Utilizamos una subrutina sencilla para transferir todos los datos necesarios para representar un dibujo (es decir, el *display file* y el fichero de atributos). Estos datos se encontrarán en algún otro lugar de la memoria, diferente de las localizaciones de memoria de la pantalla. Así podemos conseguir un método rápido de intercambio de imágenes. En un Spectrum 48K hay memoria suficiente como para mantener simultáneamente cinco imágenes alternativas o *encuadres*. Podemos utilizar el programa del listado 13.5 para construir cinco subrutinas en código máquina en las posiciones 30100, 30200, 30300, 30400 y 30500 con el fin de transferir estas imágenes en la pantalla. Supongamos que disponemos de cinco diagramas almacenados en cinta. Podemos cargarlos en el ordenador (LOAD) colocándolos en sitios apropiados de memoria de manera que puedan ser llevados alternativamente a la pantalla. Este método puede utilizarse para ilustrar una conferencia o charla de negocios con unas "diapositivas", y permite cambiar casi instantáneamente de una a otra. Naturalmente, una vez presentadas las cinco diapositivas, deberemos esperar hasta que se carguen las cinco siguientes. Esta operación puede demorarse unos cinco minutos desde cinta, pero menos de diez segundos desde disco.

Pulsando cualquier tecla entre "1" y "5" mientras utilizamos la subrutina "proyección" (listado 13.5) aparecerá en pantalla el dibujo solicitado.

Listado 13.5

```
 100 REM cargador para rutinas e
n codigo maquina
 110 CLEAR 29999: DIM F(5)
 120 FOR I =1  TO 5: RESTORE
 130 LET F(I)=120+27*(I-1)
 140 FOR J = 0 TO 11
 150 READ A: POKE 30000+I*100+J,
A
 160 NEXT J
 170 NEXT I

 200 REM transferancia de datos
del cuadro iesimo a la pantalla
```

```
 210 DATA 17,0,64            : RE
M  LD        DE,16384
 220 DATA 33,0,F(I)          : RE
M  LD        HL,FRAME(I)
 230 DATA 1,0,27             : RE
M  LD        BC,27*256
 240 DATA 237,176            : RE
M  LDIR
 250 DATA 201                : RE
M  RET

 300 REM carga los cuadros
 310 FOR J =1  TO 10: BEEP 0.1,3
0: PAUSE 5: NEXT J
 320 CLS : FOR I = 0 TO 4
 330 LOAD ""CODE (120+I*27)*256,
27*256
 340 NEXT I
 350 FOR J = 1 TO 10: BEEP 0.1,3
0: PAUSE 5: NEXT J

 400 REM proyeccion tipo diaposi
tiva
 410 IF INKEY$ <>"" THEN  GO TO
410
 420 LET A$ =INKEY$: IF A$="" TH
EN  GO TO 420
 430 IF A$ =CHR$ 13 THEN  GO TO
310
 440 IF A$ = "m" THEN  GO TO 500
 450 IF A$ >= "1" AND A$<="5" TH
EN  LET a=USR (30000+VAL A$*100)
 460 GO TO 410

 500  REM proyeccion tipo cine
 510 FOR I = 30100 TO 30500 STEP
 100
 520 LET A = USR I
 530 IF INKEY$ = "s" THEN  GO TO
 400
 540 NEXT I: GO TO 510
```

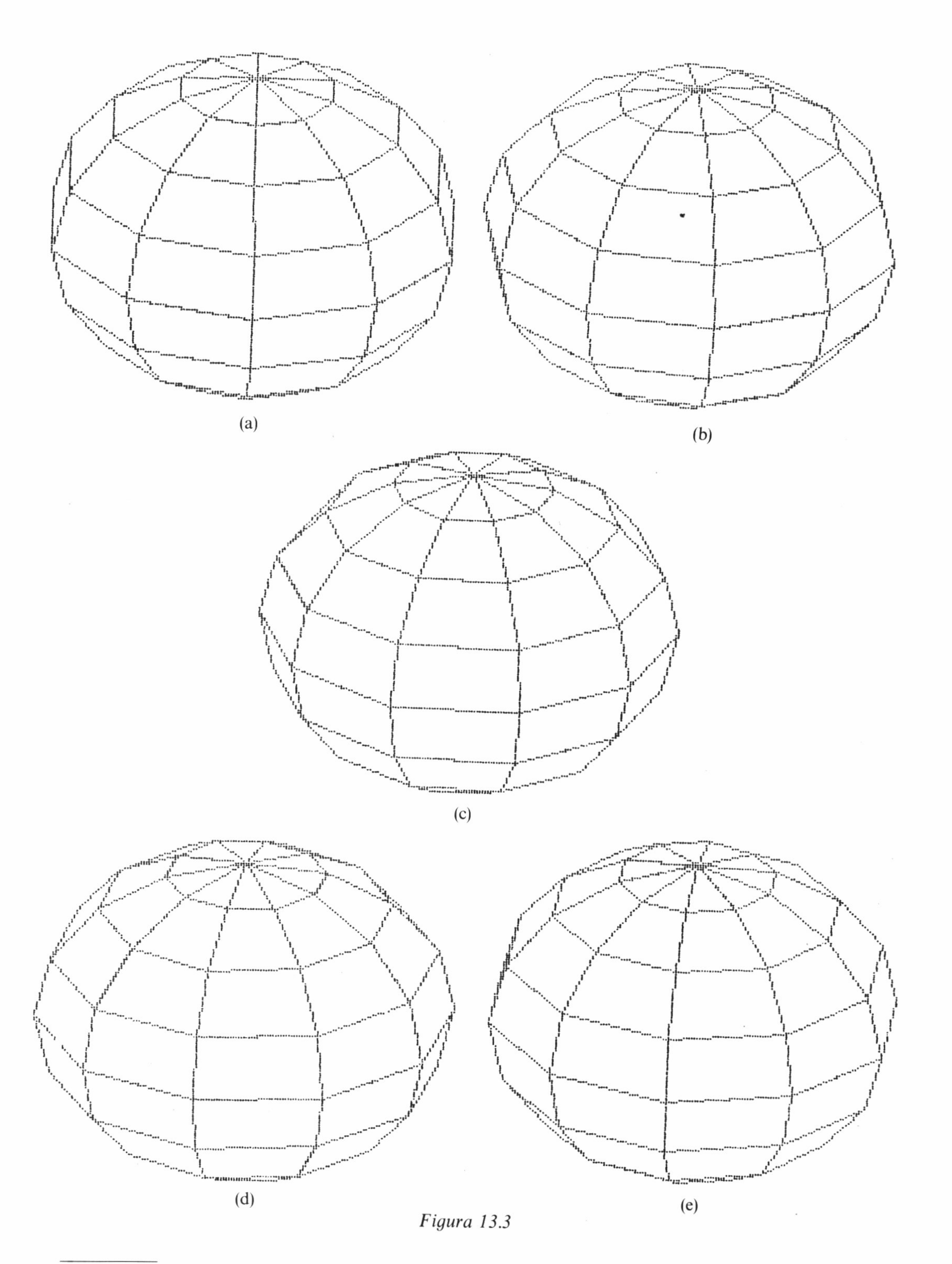


Figura 13.3

Podemos utilizar esta idea para producir "cine". Por ejemplo, observemos la figura 13.3 que muestra cinco encuadres producidos por las subrutinas de tres dimensiones del capítulo 11. Se trata de varias vistas del mismo esferoide creadas utilizando HORIZ = 3.2, VERT = 2.2, NUMH = 10, NUMV = 8, y PHI = 0.4*PI*I/NUMH, donde $0 \leqslant I \leqslant 4$. Podemos hacer "moverse" este objeto, el cual está observado desde (1, 2, 3) hacia (0, 0, 0) presentando en pantalla secuencialmente los cinco encuadres en rápida sucesión. Este conjunto de diagramas está incluido en la cinta de datos.

Una vez finalizado el programa "cine" o "proyección", deberá teclear CLEAR 65367, ya que de otro modo la próxima vez que cargue un programa (LOAD) el Spectrum responderá "Out of Memory".

Ejercicio 13.2

Dibuje una vista en perspectiva de un cubo en el encuadre 1 y la misma vista, utilizando un algoritmo de líneas ocultas, en la vista 2. Produzca una "película" consistente en estos dos encuadres únicamente; se deberán ver las aristas visibles del cubo fijas, mientras que las aristas ocultas aparecerán y desaparecerán intermitentemente. Construya una "película" en la que alguna de las "estrellas" del capítulo 12 dé la impresión de girar.

Scrolling *(ordenadores de 16 y 48K)*

La utilización de código máquina para trabajos más complicados necesita bastantes más conocimientos y un largo tiempo de estudio. Aconsejamos consultar un buen libro sobre la materia, preferiblemente uno dedicado específicamente a Spectrum. Para aquellos que deseen realizar manipulaciones más complejas de la pantalla en código máquina, daremos un nuevo ejemplo. El listado 13.6 es un programa que realiza el "enrollado" *(scroll)* de la pantalla de arriba a abajo. El problema que se nos presenta es el ya comentado de estar los datos del *display file* divididos en líneas de 32 bytes. Para poder mover estos datos con eficiencia necesitamos saber las direcciones de comienzo de estas líneas. El cálculo de direcciones en una subrutina de código máquina necesita un gran esfuerzo de programación y lentifica la ejecución. En su lugar utilizamos una *tabla de consulta.* A efectos de código máquina, una dirección se compone de 16 bits almacenados en dos posiciones de ocho bits cada una, llamadas el *byte alto* y el *byte bajo.* Todo lo que tenemos que hacer para saber la dirección de una línea es simplemente mirar estas dos mitades en las tablas. Utilizamos lenguaje BASIC para calcular las tablas y las introducimos en memoria por medio de instrucciones POKE. Estas tablas, así como el propio código máquina, pueden almacenarse en cinta y volverse a cargar si es necesario. El listado 13.6 da el programa en código máquina, el cargador BASIC y el constructor de la tabla de direcciones. El listado 13.7 muestra una subrutina BASIC equivalente al programa. Obsérvese que la subrutina BASIC supone la existencia de la misma tabla utilizada por el código máquina.

Ejercicio 13.3

Escriba una subrutina en código máquina que modifique el fichero de atributos en memoria, una fila cada vez. Escriba a continuación un programa BASIC que llame a dicha subrutina siempre que el área de gráficos ascienda ocho líneas.

Listado 13.6

```
  10 REM   cargador para rutina
en codigo maquina
  20 CLEAR 31999
  30 FOR I=0 TO 74
  40 READ A: POKE 32000+I,A
  50 NEXT I

 100 REM desplazamiento de panta
lla hacia abajo (zona graficos)
 110 DATA 1,175,0
: REM LD    BC,175
 120 DATA 221,33,75,125
: REM LD    IX,LOBYTE
 130 DATA 221,9
: REM ADD   IX,BC
 140 DATA 221,110,0
: REM LD    L,[IX+0]
 150 DATA 221,33,251,125
: REM LD    IX,HIBYTE
 160 DATA 221,9
: REM ADD   IX,BC
 170 DATA 221,102,0
: REM LD    H,[IX+0]
 180 DATA 17,171,126
: REM LD    DE,TEMP
 190 DATA 205,67,125
: REM CALL  MOVE
 200 DATA 221,33,75,125
: REM LD    IX,LOBYTE
 210 DATA 221,9
: REM ADD   IX,BC
 220 DATA 221,94,0
: REM LD    E,[IX+0]
 230 DATA 221,110,-1
: REM LD    L,[IX-1]
```

```
 240 DATA 221,33,251,125
: REM LD     IX,HIBYTE
 250 DATA 221,9
: REM ADD    IX,BC
 260 DATA 221,86,0
: REM LD     D,[IX+0]
 270 DATA 221,102,-1
: REM LD     H,[IX-1]
 280 DATA 205,67,125
: REM CALL   MOVE
 290 DATA 13
: REM DEC    C

 300 DATA 32,226
: REM JR     NZ,LINE
 310 DATA 17,0,64
: REM LD     DE,16384
 320 DATA 33,171,126
: REM LD     HL,TEMP
 330 DATA 205,67,125
: REM CALL   MOVE
 340 DATA 201
: REM RET
 350 DATA 197
: REM PUSH   BC
 360 DATA 1,32,0
: REM LD     BC,32
 370 DATA 237,176
: REM LDIR
 380 DATA 193
: REM POP    BC
 390 DATA 201
: REM RET

 400 REM construye tablas de dir
ecciones de lineas de pantalla
 410 FOR I=0 TO 21: FOR J=0 TO 7
 420 LET A=16384+INT (I/8)*2048+
(I-INT (I/8)*8)*32
 430 LET H=INT (A/256): LET L=A-
H*256
 440 POKE 32075+I*8+J,L: POKE 32
251+I*8+J,H+J
 450 NEXT J: NEXT I

 500 REM programa principal
```

```
 510 LIST 380
 520 IF INKEY$<>"" THEN  LET A=U
SR 32000
 530 GO TO 520
```

Listado 13.7

```
  10 REM version en basic del de
splazamiento de pantalla hacia
abajo (zona graficos)
  20 CLEAR 31999
  30 GO TO 400

 100 REM desplazamiento de panta
lla hacia abajo con repeticion
 110 LET C=175: LET B=0: LET BC=
C+B*256
 120 LET IXLO=75: LET IXHI=125:
LET IX=IXLO+IXHI*256
 130 LET IX=IX+BC
 140 LET L=PEEK (IX+0)
 150 LET IXLO=251: LET IXHI=125:
 LET IX=IXLO+IXHI*256
 160 LET IX=IX+BC
 170 LET H=PEEK (IX+0): LET HL=L
+H*256
 180 LET E=171: LET D=126: LET D
E=E+D*256
 190 GO SUB 350
 200 LET IXLO=75: LET IXHI=125:
LET IX=IXLO+IXHI*256
 210 LET IX=IX+BC
 220 LET E=PEEK (IX+0)
 230 LET L=PEEK (IX-1)
 240 LET IXLO=251: LET IXHI=125:
 LET IX=IXLO+IXHI*256
 250 LET IX=IX+BC
 260 LET D=PEEK (IX+0): LET DE=E
+D*256
 270 LET H=PEEK (IX-1): LET HL=L
+H*256
 280 GO SUB 350
```

```
 290 LET C=C-1: LET BC=C+B*256
 300 IF C<>0 THEN  GO TO 200
 310 LET E=0: LET D=64: LET DE=E
+256*D
 320 LET L=171: LET H=126: LET H
L=L+H*256
 330 GO SUB 350
 340 RETURN
 350 LET S=BC
 360 LET BC=32
 370 LET A=PEEK HL: POKE DE,A: L
ET DE=DE+1: LET HL=HL+1: LET BC=
BC-1: IF BC<>0 THEN  GO TO 370
 380 LET BC=S
 390 RETURN

 400 REM construye tablas de dir
ecciones de lineas de pantalla
 410 FOR I=0 TO 21: FOR J=0 TO 7
 420 LET A=16384+INT (I/8)*2048+
(I-INT (I/8)*8)*32
 430 LET H=INT (A/256): LET L=A-
H*256
 440 POKE 32075+I*8+J,L: POKE 32
251+I*8+J,H+J
 450 NEXT J: NEXT I

 500 REM programa principal

 510 LIST 380
 520 IF INKEY$<>"" THEN  GO SUB
100
 530 GO TO 520
```

Estructura BASIC (renumerado y borrado)

El desarrollo de programas modulares conduce a menudo a situaciones en que las subrutinas están enlazadas de tal manera que no queda sitio para líneas extra o alteraciones. En un momento como ése querríamos poder *renumerar* las líneas automáticamente o quizá *borrar* secciones del programa. Estas dos funciones serían también útiles cuando intentamos fundir programas por medio de MERGE con el fin de crear nuevas subrutinas. Si queremos realizar estas tareas, debemos inspeccionar cuidadosamente la forma en que nuestros programas BASIC se almacenan en memoria.

Los programas BASIC están almacenados línea por línea en la memoria comenzando en la dirección 23755. Lo primero que se almacena en cada línea es el propio número de la línea (16 bits), que ocupa dos posiciones y queda almacenado como byte alto seguido de byte bajo. En todas las demás ocasiones, los valores se almacenan en la forma estándar de Z80, es decir, con formato bajo-alto. Como ejemplo, supongamos que introducimos la línea 10 REM. Si tecleamos

PRINT PEEK 23755, PEEK 23756

comprobaremos que en dichas posiciones se encuentran almacenados un 0 y un 10. Si eliminamos la línea 10 e introducimos otra línea numerada al comienzo del programa, observaremos que la representación de este nuevo número está almacenado de nuevo en 23755 y 23756. Los dos siguientes bytes dan la longitud de la línea (podemos utilizarlos para encontrar el comienzo de la siguiente línea de programa sin necesidad de recorrer la línea completa). A continuación aparece el texto real de la línea BASIC, ocupando cada carácter u orden una posición de memoria. El final de línea se marca con CODE 13 (o ENTER). Detrás de cada número dado en notación decimal y almacenado como caracteres hay un CODE 14 (o NUMERO: véase Manual BASIC del Spectrum) seguido por la traducción binaria del número, que ocupa cinco bytes. Los enteros se almacenan de forma sencilla, tal como se explica también en el Manual de Spectrum. Utilizando esta información sobre el contenido de la memoria, podemos escribir un programa que liste programas, incluyendo a sí mismo, a base de examinar la memoria. Este tipo de programa autolistante (listado 13.8) puede ser de gran ayuda cuando formateamos listados en los cuales todas las sentencias comienzan en nueva línea, o cuando deseamos que aparezcan los códigos de control.

Listado 13.8

```
 100 REM programa autolistante
 110 LET I=23755: CLS
 120 LET LINE=256*PEEK I+PEEK (I
+1): IF LINE>9999.5 THEN  STOP
 130 PRINT " ";LINE;
 140 LET I=I+2: PRINT  PAPER 5;P
EEK I;",";PEEK (I+1);" ";
 150 LET LENGTH=PEEK I+256*PEEK
(I+1)
 160 LET NUM=0: LET I=I+1
 170 FOR J=1 TO LENGTH: LET I=I+
1
 180 LET P=PEEK I: IF P<32 AND N
UM<=0 THEN  LET NUM=1
```

```
 190 IF P=13 THEN  PRINT  PAPER
4;P: PRINT : GO TO 230
 200 IF P=14 THEN  LET NUM=6
 210 IF NUM>0 THEN  PRINT  PAPER
6;P;",";: GO TO 230
 220 PRINT CHR$ P;
 230 LET NUM=NUM-1
 240 NEXT J
 250 LET I=I+1: GO TO 120
```

Para renumerar líneas, deberemos cambiar los números de línea almacenados en memoria. Esto es cierto en tanto en cuanto las líneas no se aparten de una secuencia numérica. Desgraciadamente debemos tener en cuenta también las instrucciones como GO TO o GO SUB, etc. Deberemos buscar todas las apariciones de GO SUB, GO TO, RESTORE y RUN en el programa y, si se hallan seguidas por un entero simple, comprobar cuál es el nuevo número de línea que corresponde. En tales casos, deberemos cambiar los caracteres numéricos del número así como la representación entera del mismo.

Si deseamos borrar una serie de líneas, todo lo tenemos que hacer es extender la longitud de la línea anterior hasta cubrir las no deseadas, de manera que dicha área queda anulada. A continuación se restablece la longitud correcta de la línea editando y volviendo a introducir dicha línea. El resto del programa se mueve por la memoria colocándose a continuación del nuevo final de la línea. El listado 13.9 presenta dos subrutinas que realizan estas tareas: la subrutina renumeración se inicia con RUN 9900 y la de borrado con RUN 9970.

Listado 13.9

```
9900 INPUT "Renumerar lineas ";L
OW;" a ";UP,"Empezando por ";NEW
'"en incrementos de ";STEP
9901 LET X=9900: IF LOW>X OR UP>
X OR LOW>UP THEN  GO TO 9900
9902 IF NEW>X OR NEW<1 OR STEP>X
 OR STEP<1 THEN  GO TO 9900
9903 POKE 23692,-1: PRINT AT 19,
0;"Renumerar lineas ";LOW;" a ";
UP,"Empezando por ";NEW,"en incr
ementos de ";STEP'"Espere, por f
avor"
9904 LET START=23755: LET SCREEN
=16384: LET HI=256
9905 LET A=START: LET B=SCREEN
```

```
9906 LET H=PEEK A: LET L=PEEK (A
+1)
9907 POKE B,H: POKE B+1,L
9908 LET A=A+2: LET B=B+2
9909 LET LEN=PEEK A+PEEK (A+1)*H
I: LET A=A+LEN+2
9910 IF H*HI+L<X THEN  GO TO 990
6
9911 LET A=START: LET B=SCREEN:
LET W=NEW
9912 LET H=PEEK A: LET L=PEEK (A
+1)
9913 LET N=HI*H+L: IF N<LOW OR N
>UP THEN  GO TO 9918
9914 IF W>X THEN  GO TO 9955
9915 LET H=INT (W/HI): LET L=W-H
*HI
9916 POKE A,H: POKE A+1,L
9917 LET W=W+STEP
9918 LET A=A+2
9919 LET LEN=PEEK A+PEEK (A+1)*H
A: LET A=A+LEN+2
9920 IF N<X THEN  GO TO 9912
9921 LET A=START: LET B=SCREEN
9922 LET H=PEEK A: LET L=PEEK (A
+1): LET NN=PEEK B*HI+PEEK (B+1)
9923 LET N=H*HI+L: IF N=X THEN
STOP
9924 PRINT AT 21,0;"Comprobando
";NN;"=nueva linea ";N
9925 LET A=A+2: LET W=A+2
9926 LET LEN=PEEK A+PEEK (A+1)*H
I: LET A=A+LEN+2
9927 LET T=PEEK W
9928 IF T=CODE (" GO TO ") OR T=
CODE (" GO SUB ") OR T=CODE (" R
UN ") OR T=CODE (" RESTORE ") TH
EN  GO SUB 9932
9929 IF T=14 THEN  LET W=W+5
9930 LET W=W+1: IF W<A THEN  GO
TO 9927
9931 LET B=B+2: GO TO 9922
9932 LET V=W+1: LET A$=""
9933 LET S=PEEK V: IF S<>32 AND
S<>14 AND NOT (S>=48 AND S<=57)
```

```
THEN  RETURN
9934 IF S<>14 THEN  LET V=V+1: L
ET A$=A$+CHR$ S: GO TO 9933
9935 LET V=V+3: LET AT=PEEK V+PE
EK (V+1)*HI
9936 IF AT<LOW OR AT>UP THEN  RE
TURN
9937 GO SUB 9943: LET H=INT (NAT
/HI): LET L=NAT-H*HI
9938 LET B$=STR$ NAT: IF LEN A$<
>LEN B$ THEN  GO SUB 9951
9939 POKE V,L: POKE V+1,H
9940 FOR I=1 TO LEN B$
9941 POKE W+I,CODE B$(I): NEXT I
9942 RETURN
9943 LET C=START: LET D=SCREEN
9944 LET H=PEEK D: LET L=PEEK (D
+1)
9945 LET E=PEEK C*HI+PEEK (C+1)
9946 IF E>=X THEN  LET NAT=0: RE
TURN
9947 IF H*HI+L=AT THEN  LET NAT=
E: RETURN
9948 LET C=C+2: LET D=D+2
9949 LET LEN=PEEK C+PEEK (C+1)*H
I: LET C=C+LEN+2
9950 GO TO 9944
9951 LET DIFF=LEN A$-LEN B$
9952 IF DIFF>0 THEN  FOR I=1 TO
DIFF: LET B$=B$+" ": NEXT I: RET
URN
9953 PRINT AT 16,0;"No hay sitio
 en ";CHR$ T;AT,"en la linea ";N
N;".","Teclea EDIT y pon ";-DIFF
;" espacio(s)","a la etiqueta, l
uego reejecuta","el programa."
9954 LET H=INT (NN/HI): POKE 236
26,H: POKE 23625,NN-H*HI
9955 LET A=START: LET B=SCREEN
9956 LET H=PEEK A: LET L=PEEK (A
+1)
9957 IF H*HI+L=X THEN  PRINT AT
0,0;"Renumeracion abortada": STO
P
9958 POKE A,PEEK B: POKE A+1,PEE
```

```
K (B+1)
9959 LET A=A+2: LET B=B+2
9960 LET len=PEEK A+PEEK (A+1)*H
I: LET A=A+LEN+2
9961 GO TO 9956
9970 INPUT "Borrar lineas ";LOW;
" a ";UP: IF LOW<2 OR LOW>=UP TH
EN  GO TO 9970
9971 PRINT AT 20,0;"Borrar linea
s ";LOW;" a ";UP,"Espere, por fa
vor"
9972 LET START=23755: LET SCREEN
=16384: LET HI=256
9973 LET A=START: LET B=SCREEN
9974 LET H=PEEK A: LET L=PEEK (A
+1)
9975 IF H*HI+L>=LOW THEN  PRINT
AT 0,0;"Por favor, teclee la lin
ea ",LOW-1;" REM y reejecute el
programa": STOP
9976 LET H=PEEK A: LET L=PEEK (A
+1)
9977 IF H*HI+L>=LOW THEN  GO TO
9981
9978 LET NN=H*HI+L: LET LAST=A:
LET A=A+2
9979 LET LEN=PEEK A+PEEK (A+1)*H
I: LET A=A+LEN+2
9980 GO TO 9976
9981 LET LAST=LAST+2
9982 LET LONG=PEEK LAST+PEEK (LA
ST+1)*HI
9983 IF H*HI+L>UP THEN  PRINT AT
 0,0;"Ninguna linea en el margen
 dado": STOP
9984 LET H=PEEK A: LET L=PEEK (A
+1): IF H*HI+L>UP THEN  GO TO 99
90
9985 LET A=A+2: LET LONG=LONG+2
9986 LET LEN=PEEK A+PEEK (A+1)*H
I: LET A=A+LEN+2: LET LONG=LONG+
LEN+2
9987 IF H*HI+L<>UP THEN  GO TO 9
984
9990 LET H=INT (LONG/HI): POKE L
```

```
AST+1,H: POKE LAST,LONG-H*HI
9995 LET H=INT (NN/HI): POKE 236
26,h: POKE 23625,NN-H*HI
9999 PRINT AT 0,0;"Teclee EDIT y
 ENTER": STOP
```

Ejercicio 13.4

Escriba una subrutina que busque en la memoria BASIC y encuentre una secuencia determinada de códigos CODE. Utilice el método mostrado en el programa de borrado anterior para asignar la variable del sistema E PPC (que controla el cursor de edición: véase Manual de programación BASIC de Spectrum) al número de línea en que se encuentra la sentencia.

Estructura BASIC (programas eficientes)

Al programar en BASIC podemos utilizar los conocimientos adquiridos acerca de la forma de almacenamiento de las líneas para conseguir que nuestros programas sean más eficientes, tanto en términos de espacio utilizado como en velocidad de ejecución. Cada vez que aparece en una línea de programa un número de forma explícita, el mismo viene seguido por seis bytes de códigos numéricos, de tal forma que no se necesite transformar la cadena de dígitos en su forma binaria cada vez que se ejecute la línea. En su lugar podemos asignar el número requerido a una variable, y utilizar el nombre de la variable en lugar de su valor de manera que el espacio necesario será ahora únicamente el número de bytes que contenga el nombre de la variable. Para asignar el valor deberemos utilizar tres códigos extra, ":", "LET" y "=", así como el nombre de la variable, y para acceder al valor deberemos utilizar dicho nombre. Si el valor se va a utilizar con cierta frecuencia, el ahorro de espacio puede ser considerable. Observemos las dos primeras sentencias del ejemplo siguiente:

```
10 PRINT AT 1,1
20 LET A=1:PRINT AT A,A
30 PRINT AT 1,1
40 PRINT AT A,A
```

El texto BASIC de la línea 10 necesita 17 bytes de almacenamiento: "PRINT", "AT", "1", 14, 0, 0, 1, 0, 0, ",", "1", 14, 0,0, 1, 0, 0.

Por el contrario, la línea 20 necesita únicamente 16 bytes: "LET", "A", "=", "1", 14, 0, 0, 1, 0, 0, ":", "PRINT", "AT", "A", ",", "A".

Observamos que la segunda línea ocupa menos espacio que la primera. Sin embargo, la segunda línea utiliza la variable A, la cual ha de almacenarse en otro lugar ocupando seis bytes de memoria. Si la sentencia debe utilizarse varias veces, como

sucede en las líneas 30 y 40, encontraremos que la línea 30 tiene una longitud de 17 bytes en tanto que la línea 40 ocupa únicamente 5 bytes. Así pues, si se va a utilizar un valor numérico más de dos veces en un programa, conviene almacenarlo como variable para ahorrar espacio.

También podemos ahorrar espacio y aumentar la velocidad empaquetando el mayor número posible de sentencias en una línea, aunque como contrapartida perdemos legibilidad en el programa. Cada nueva línea requiere cinco bytes de memoria: uno para CODE 13 al final de la línea anterior, dos para el número de la nueva línea y dos para su longitud. Sin embargo un símbolo dos puntos como separación de sentencias necesita únicamente un byte: la cantidad de espacio ahorrado puede ser considerable. Por ejemplo, supongamos un programa con 90 sentencias (bastante pequeño como promedio): si las sentencias están separadas inicialmente línea a línea y las reescribimos de manera que ocupamos tres sentencias por línea, habremos ahorrado 60*(5 − 1) = 240 bytes. En un programa grande es bastante fácil ahorrar más de 1K de memoria de esta forma. La utilización de menos líneas hace asimismo a la máquina ejecutar con mayor facilidad instrucciones del tipo GO TO o GO SUB, ya que le lleva menos tiempo el rastreo de la línea de destino. Se consigue la máxima eficiencia colocando todas las subrutinas y funciones cerca del comienzo del programa, ordenadas de modo que la más utilizada vaya en primer lugar. A menos que se vaya a realizar un número realmente grande llamadas a una subrutina, generalmente es más conveniente almacenarlas en un orden lógico.

Ejercicio 13.5

Reescriba las subrutinas gráficas y/o los programas de juego para utilizar menos espacio y, a ser posible, ejecutarse con mayor rapidez.

Display *sincronizado*

Se puede conseguir un interesante efecto BORDER utilizando el comando PAUSE al realizar dibujos en el Spectrum. Una imagen de televisión se dibuja completamente cada 1/50 de segundo (1/60 en Estados Unidos) y el comando PAUSE utiliza múltiplos de este mismo intervalo de tiempo. La orden PAUSE 1 no siempre retrasa la ejecución durante 1/50 de segundo: de hecho, PAUSE permanece únicamente hasta el comienzo de la siguiente imagen. Este hecho proporciona un método de comenzar instrucciones en un momento determinado con respecto al refresco de la pantalla. Si cambiamos el color BORDER durante el dibujo de una escena, en la misma imagen se coloreará únicamente la parte superior del color asignado a BORDER mientras que el resto permanecerá de otro color. Obviamente, podemos utilizar PAUSE para esperar al comienzo del refresco de la nueva imagen y repetir el proceso. El resultado es un dibujo estático con un BORDER de distintos colores. Hemos observado este efecto al ejecutar sentencias SAVE y LOAD, las cuales utilizan una subrutina en código máquina que realiza un proceso equivalente, en el cual se intercambian rojo/cian y amarillo/azul en la parte externa (BORDER) de la pantalla.

Por ejemplo, consideremos este comienzo de un programa largo:

```
1 GO TO 10
2 PAUSE 1:BORDER 7:BORDER 2:BORDER 6:BORDER 4:
BORDER 5:BORDER 1:BORDER 3:BORDER 7:GO TO 2
```

El programa finaliza con un comando GO TO 2 que produce un efecto de arco iris en el BORDER tras la construcción del diagrama. Obsérvese que, a menos que la línea se coloque cerca del comienzo del programa, la sentencia GO TO llevará demasiado tiempo para ejecutarse, y perderemos el comienzo de la siguiente imagen.

Ejercicio 13.6

Escriba un programa de una sola línea que cambie entre colores rojo y cian en BORDER sin utilizar PAUSE. Inserte símbolos ":" extra (no otras sentencias, simplemente los dos puntos) entre las sentencias hasta conseguir que el tiempo de ejecución de la línea sea exactamente 1/50 de segundo: es decir, que los colores permanezcan estacionarios. (*Clave:* cuéntese un comando BORDER como 10, un GO TO como 13 y un dos puntos como 1; intente que la suma total del valor de la línea, contada de esa forma, sea alrededor de 160).

Programas completos

I. Listado 13.1. Datos necesarios: 10 conjuntos de pares de coordenadas X/Y. La pantalla mostrará una tabla puesta a cero de esos valores, seguida de "Fin". Disponemos de un cursor, que apunta a la primera fila, y puede moverse con "6" hacia abajo y "7" hacia arriba (en mayúsculas). Una vez que esté en la fila deseada, teclee "EDIT" ("1" mayúscula) y la máquina preguntará las coordenadas X/Y de *pixel* para dicha entrada de la tabla. Cuando haya terminado, mueva el cursor hasta "Fin"; pulse a continuación EDIT y teclee "s"(sí). El programa dibujará a continuación un polígono uniendo los 10 puntos de coordenadas.

II. Listado 13.2. Datos necesarios: entrada numérica para el eventual sistema de fichero. Teclee cualquier número entre "1" y "6". Si utiliza "5" ó "6" la máquina pregunta si realmente desea realizar la operación: teclee "s"(sí) o "n"(no). El programa termina con BREAK.

III. Listado 13.3. No necesita datos: el programa termina con BREAK.

IV. Listado 13.4. Pulse cualquier tecla cuando se le solicite. No necesita datos: el programa termina con BREAK.

V. Listado 13.5. Tras escuchar el sonido (BEEP) el programa solicita cinco dibujos para ser cargados desde cinta. Teclee "1", "2", "3", "4" ó "5" para presentación del encuadre correspondiente; "m" para "cine" y "s" para detener el movimiento.

VI. Listado 13.6. Cuando la pantalla esté llena, deje pulsada cualquier tecla para que el *scroll* continúe. Utilice BREAK para detener.

VII. Listado 13.7 Versión BASIC de VI. ¡Muy lento!

VIII. Listado 13.8. No necesita datos.

IX. Listados 13.7 y 13.9. Teclee RUN 9900 para renumerar y RUN 9970 para borrar. Ejemplo: editar listado 13.7

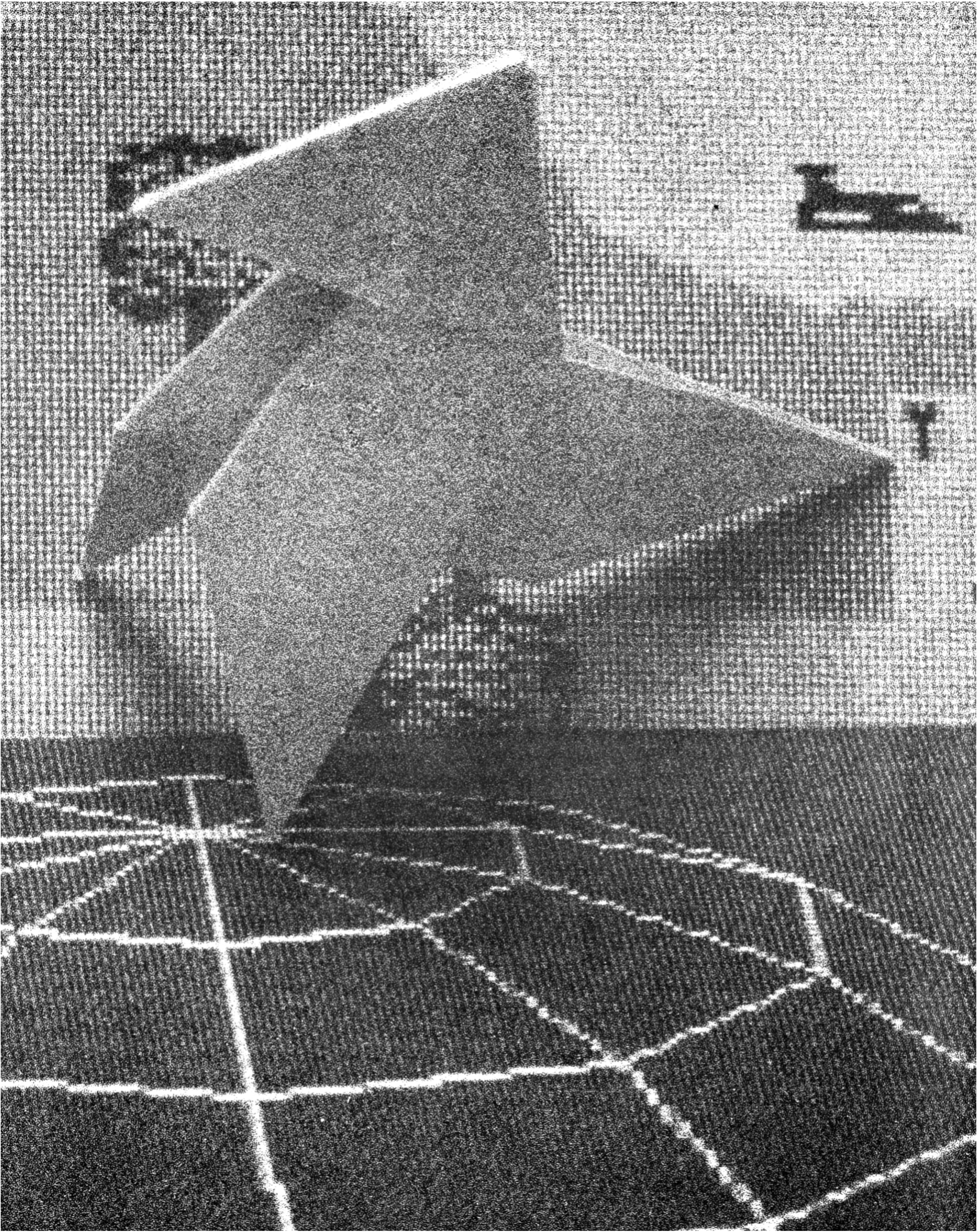
RUN 9900
Renumerar líneas 10 a 30
Empezando por 60 en incrementos de 5

Ahora liste el programa (LIST) para comprobar los cambios

RUN 9970
Borrar líneas 60 a 99

Atención: debe existir al menos una línea sobre la sección a borrar

RUN 9970
Borrar líneas 65 a 99
EDIT ("1" en mayúsculas)
LIST



14
Un ejemplo detallado de un videojuego

En este capítulo exploraremos los límites de la programación BASIC para realizar videojuegos animados. Nuestra experiencia indica que los juegos escritos en BASIC son caros, y en general el interés de los jugadores decae rápidamente. En el momento que los usuarios alcanzan resultados razonables con sus propios programas, prefieren escribirlos ellos mismos, invirtiendo su dinero exclusivamente en juegos sofisticados realizados en código máquina. El listado 14.1 es un ejemplo del tipo de juego que puede aspirar a escribir un programador BASIC muy competente sin echar mano a rutinas en código máquina. El juego DEFENSA DE LA ISLA (sólo para ordenadores de 48K) es un típico juego de "marcianitos", pero las técnicas que discutimos aquí pueden aplicarse igualmente a juegos de "raqueta y pelota" (por ejemplo, TENIS) o a juegos "tácticos" (por ejemplo, COMECOCOS).

Presentación del juego

Hemos dibujado en la pantalla una escena que consta de una isla verde con una pequeña colina, mar azul oscuro, y cielo azul claro con un sol amarillo y una nube blanca. En la isla hay tres árboles, así como una zona asfaltada de color azul claro en la cual hemos colocado una tienda. El juego comienza con un hombre que sale de la tienda y se dirige a un emplazamiento antiaéreo rodeado de sacos de arenas, en el que deposita 20 proyectiles, regresando a continuación a la tienda. En ese momento, aparecen aviones enemigos saliendo del sol, que vuelan sobre la colina y bombardean el campo. Nuestro objetivo es detener el ataque disparando a los aviones. Por cada avión que ataca pueden suceder dos cosas: o bien el aeroplano es derribado, o consigue un impacto en nuestro campo. Cada impacto disminuye el tamaño de

la tienda. Si el enemigo consigue arrojar 14 bombas sobre la tienda, ésta desaparece, y el siguiente avión bombardea nuestra arma antiaérea. Con la destrucción del antiaéreo, el borde de la pantalla cambia de color repetidamente, y el juego recomienza. Después del bombardeo, cada avión vuela a través de la nube, pasa por encima del antiaéreo y abandona la escena por la izquierda. Cada 20 aviones recibimos un nuevo suministro de munición. Si perdemos 3 antiaéreos, la pantalla se vuelve roja y el juego finaliza, pudiendo recomenzar desde el principio.

El antiaéreo tiene siete posibles posiciones de disparo, accesibles por las teclas "1" a "7", y dispara un misil cuando se pulsa "x". Debido a las restricciones de velocidad se permite un solo avión y un solo misil en pantalla en cada momento.

La mejor manera de comprender el resto de las explicaciones de este capítulo es cargar y ejecutar el listado 14.1 que va incluido en la cinta. El programa solicitará que se cargue (LOAD) un conjunto de caracteres especiales, llamado "video" (para dibujar el avión, la tienda, el hombre, etc.), y a continuación "escena" (para preparar el fondo); estos dos ficheros deberán cargarse antes de comenzar el juego. El listado está comentado con profusión, de manera que no es necesario insistir mucho aquí en los detalles. En su lugar, estudiaremos un sistema general de construcción de este tipo de juegos, y describiremos métodos para resolver los problemas típicos que aparecen al prepararlos.

Listado 14.1

```
   9 REM Inicializa las rutinas
para permitir el uso del juego d
e caracteres alternativo
  10 CLEAR 62294: INK 0: PAPER 7
: BORDER 7: FLASH 0
  20 DIM S(6): FOR I=1 TO 6: REA
D S(I): NEXT I
  30 DATA 15360,62039,62807,6357
5,64343,64848
  40 LET set=50: LET S=1: GO SUB
 set: GO TO 200

  50 REM juego/cambio al juego S
  51 REM Datos de entrada S
  60 LET HI=INT (S(S)/256): LET
LO=S(S)-HI*256
  70 POKE 23606,LO: POKE 23607,H
I: RETURN

  79 REM Carga los caracteres de
l juego 2
  80 REM cargacaracteres
```

```
  90 LET N$="videojuego"
 100 LET S=2
 110 INPUT ("   CARGANDO "+N$+CH
R$ 6+"Arranca el casete y da ENT
ER."); LINE X$
 120 LOAD N$ CODE (S(S)+256),768
: RETURN

 200 REM programa principal
 209 REM Inicializa los punteros
 de las rutinas
 210 LET charload=80: LET load=4
600: LET create=5000: LET credit
=5500: LET char=5600
 220 LET keyboard=500: LET camp=
700: LET status=800: LET plane=9
80
 230 LET reload=1500: LET ammo=1
900: LET missile=2200: LET explo
de=2500
 240 LET bombcamp=3500: LET bomb
gun=3000: LET hiscore=5700: LET
HSC=0
 249 REM Carga el juego de carac
teres si en necesario
 250 IF PEEK 62303<>110 THEN  GO
 SUB charload
 258 REM Carga el escenario e im
prime los nombres
 260 GO SUB load: GO SUB create:
 GO SUB credit
 269 REM Hace una copia de los a
tributos de color de la pantalla
 270 DIM A(704): FOR I=1 TO 704:
 LET A(I)=PEEK (22527+I): NEXT I
 279 REM Prepara los literales p
ara el uso de las rutinas de pan
talla
 280 LET S$="  PUNTOS    ": LET
B$="  BASES       "
 290 FOR I=1 TO 4: LET S$=S$+CHR
$ 8: LET B$=B$+CHR$ 8: NEXT I
 300 DIM N$(9): LET N$="
 "
```

```
 309 REM Comienzo del juego
 310 BRIGHT 1: PAPER 8: INK 8: O
VER 0
 319 REM Inicializa la puntuacio
n y el no. de bases
 320 LET SC=0: LET BS=3: GO SUB
status
 329 REM Quita la municion reman
ente y el parpadeo del antiaereo
 330 PRINT AT 17,5;"  ": PRINT A
T 18,5;"  ": PRINT AT 16,3;" "
 339 REM Restablece los atributo
s de color
 340 FOR I=1 TO 704: POKE 22527+
I,A(I): NEXT I
 349 REM renueva la puntuacion y
 la municion
 350 GO SUB camp: GO SUB status:
 GO SUB reload
 358 REM Prepara el bucle princi
pal del programa
 360 LET p=plane: LET m=missile:
 LET k=keyboard
 369 REM Comienza una oleada de
20 aviones
 370 LET AR=20
 379 REM Comienzo del bucle prin
cipal
 380 GO SUB p: OVER 1: GO SUB m:
 OVER 0: GO SUB k
 389 REM Sale del bucle
 390 IF DEAD  THEN  GO TO 420
 399 REM Si quedan aviones sigue
 en el bucle
 400 IF AR>0 THEN  GO TO 380
 409 REM Si no quedan aviones em
pieza una nueva oleada
 410 GO SUB reload: GO TO 370

 419 REM Si quedan bases usa la
siguiente
 420 INK 0: PRINT AT 12,0;"     "
: INK 8: IF BS>1 THEN  LET BS=BS
-1: GO TO 350
```

```
 429 REM El juego ha terminado,
parpadea el borde
 430 OVER 1: FOR I=1 TO 22: BEEP
 0.005,RND*30-15: BORDER RND*7
 440 FOR J=1 TO 2: PLOT 0,(22-I)
*8: DRAW  BRIGHT 1; PAPER 2; INK
 2;255,0
 450 NEXT J: NEXT I: BORDER 7
 460 FOR I=1 TO 5: GO SUB m: NEX
T I: OVER 0

 469 REM Comprueba si has alcanz
ado la maxima puntuacion y recom
ienza el juego
 470 IF SC>HSC THEN  GO SUB hisc
ore
 480 PAUSE 200: GO TO 310

 500 REM teclado
 509 REM Si no se aprieta una te
cla en el bucle se pierde la opo
rtunidad
 510 LET A$=INKEY$: IF A$="" THE
N  RETURN
 518 REM La tecla de fuego es ap
retada:
 519 REM Si quedan misiles y hay
 uno preparado se dispara
 520 IF A$="X" OR A$="x" THEN  I
F m=missile AND NOT OUT THEN  LE
T m=f: LET M$=F$: GO SUB ammo: O
VER 1: GO SUB m: OVER 0: GO TO 5
50
 529 REM Si la tecla apretada no
 esta entre 1 o 7 la ignora
 530 IF A$>"7" OR A$<"1" THEN  R
ETURN
 538 REM Calcula que rutina es u
sada si el misil es disparado en
 esta direccion
 539 REM Fija el literal usado p
or el misil y cambia la direccio
n de tiro
 540 LET f=2000+10*VAL A$: LET F
$=CHR$ (83+VAL A$)
```

```
 550 PRINT AT 16,3;F$
 560 RETURN

 700 REM campamento
 709 REM Imprime el tirador y la
 tienda
 710 PRINT AT 17,3;"z": BRIGHT 0
: BORDER 7: PRINT AT 16,23;"vw":
 PRINT AT 17,23;"xy": BRIGHT 1
 719 REM Usa la rutina del tecla
do para la posicion de tiro
 720 LET YT=14: LET HIT=0: LET D
EAD=0: LET A$="3": GO TO 540

 800 REM estado
 809 REM Usa el juego de caracte
res 1 para imprimir la puntuacio
n y las bases
 810 LET S=1: GO SUB set
 820 PRINT AT 21,0;S$;" ";SC,B$;
BS
 830 LET S=2: GO SUB set
 840 RETURN

 980 REM avion
 989 REM Fija el puntero de bomb
ardeo al principio de la cascada
 990 PRINT AT 13,5;" ": PRINT AT
 15,4;" ";: PRINT AT 17,3;"z": L
ET bombgun=3000
 999 REM Cambia el puntero, si e
s >1000 se lanza un avion
1000 LET p=991+INT (RND*11): RET
URN
1009 REM Lanza el avion, carga l
a bomba
1010 LET AR=AR-1: LET p=1020: LE
T BOMB=1: RETURN
1019 REM Mueve el avion a la der
echa
1020 PRINT AT R,C;" IJ": LET C=C
+1: IF C<10 THEN  RETURN
1029 REM Si el avion esta a 10 c
olumnas de la izquierda mueve el
 puntero hacia abajo en la casca
```

```
da
1030 LET p=1040
1039 REM Picado diagonal
1040 PRINT AT R-1,C;" ": PRINT A
T R,C;" N": LET R=R+1: LET C=C+1
: PRINT AT R,C;"n": IF C<20 THEN
  RETURN
1049 REM Si esta en la columna 2
0 mueve el puntero y borra el av
ion a la izquierda
1050 LET p=1060: PRINT AT R-1,C;
"  "
1059 REM Llama a la cascada de b
ombas las 5 veces que se usa est
a seccion
1060 GO SUB bombcamp: PRINT AT R
,C;" IJ": LET C=C+1: IF C<25 THE
N  RETURN
1069 REM Mueve el puntero hacia
abajo, comienza la curva hacia l
a nube
1070 LET p=1080
1080 PRINT AT R,C;"   ": LET R=R-
1: LET C=C+1: PRINT AT R,C;"KL":
 IF C<28 THEN  RETURN
1090 LET p=1100: PRINT AT R,C;"
": LET C=C+1
1100 PRINT AT R+1,C;" ": PRINT A
T R,C;"o": LET R=R-1: PRINT AT R
,C;"O": IF R>7 THEN  RETURN
1110 LET p=1120: PRINT AT R+1,C;
"  "
1120 PRINT AT R,C;"   ": LET R=R-
1: LET C=C-1: PRINT AT R,C;"kl":
 IF R>3 THEN  RETURN
1130 LET p=1140
1139 REM Vuela en la nube, no es
 visible
1140 PRINT AT R,C;"ij ": LET C=C
-1: IF C>11 THEN  RETURN
1150 LET p=1160: PRINT AT R,C;"
  "
1159 REM Vuela hacia el antiaere
o
1160 PRINT AT R,C;" ": LET R=R+1
```

```
: LET C=C-1: PRINT AT R,C;"M ":
PRINT AT R+1,C;"m": IF R<6 THEN
 RETURN
1170 LET p=1180
1180 PRINT AT R,C;" ": LET R=R+1
: PRINT AT R,C;"P ": PRINT AT R+
1,C;"p": IF R<8 THEN  RETURN
1189 REM Comprueba y borra algun
 posible misil perdido
1190 LET p=1200: IF X<>9 OR Y<>7
 THEN  PRINT AT 7,9;" "
1200 PRINT AT R,C;" ": LET R=R+1
: LET C=C-1: PRINT AT R,C;"M ":
PRINT AT R+1,C;"m": IF R<11 THEN
  RETURN
1210 LET p=1220: PRINT AT R,C;"
": LET R=R+1
1219 REM Ultima seccion del vuel
o, llama cada vez a la cascada d
e bombardeo
1220 GO SUB bombgun: LET C=C-1:
PRINT AT R,C;"ij ": IF C>0 THEN
 RETURN
1229 REM Avion saliendo de panta
lla
1230 LET p=1240: PRINT AT R,C;"j
 ": RETURN
1239 REM Fija la fila del avion
a 2 y el puntero al comienzo de
la cascada
1240 PRINT AT R,C;" ": LET R=2:
LET p=plane
1250 RETURN

1500 REM recarga
1509 REM Si el campamento es des
truido no renueva la municion
1510 IF HIT  THEN  RETURN
1519 REM Animacion de la figura
1520 BRIGHT 0: LET R=18
1529 REM Camina de la tienda a l
os arboles
1530 FOR C=24 TO 19 STEP -1: PRI
NT AT R,C;"S": PRINT AT R+1,C;"s
": BEEP 0.01,-15: PAUSE 5
```

```
1540 PRINT AT R,C;"R": PRINT AT
R+1,C;"r": PAUSE 3
1550 PRINT AT R,C;" ": PRINT AT
R+1,C;" ": NEXT C
1559 REM Usa sobreimpresion para
 no borrar los arboles
1560 OVER 1: PRINT AT R,18;"S":
PRINT AT R+1,18;"s": BEEP 0.01,-
15: PAUSE 5: PRINT AT R,18;"S":
PRINT AT R+1,18;"s"
1570 PRINT AT R,18;"R": PRINT AT
 R+1,18;"r": PAUSE 3: PRINT AT R
,18;"R": PRINT AT R+1,18;"r"
1579 REM Hace ruidos al desapare
cer la figura tras los arboles
1580 FOR J=1 TO 2: BEEP 0.01,-15
: PAUSE 11: NEXT J
1589 REM Muestra el pie saliendo
 tras el arbol
1590 PRINT AT R+1,15;"s": BEEP 0
.01,-15: PAUSE 5: PRINT AT R+1,1
5;"s"
1600 PRINT AT R+1,15;"r": PAUSE
3: PRINT AT R+1,15;"r"
1609 REM Muestra la figura compl
eta
1610 PRINT AT R,14;"S": PRINT AT
 R+1,14;"s": BEEP 0.01,-15: PAUS
E 5: PRINT AT R,14;"S": PRINT AT
 R+1,14;"s"
1620 PRINT AT R,14;"R": PRINT AT
 R+1,14;"r": PAUSE 3: PRINT AT R
,14;"R": PRINT AT R+1,14;"r"
1629 REM Camina del arbol a la h
ierba en la columna 8
1630 OVER 0: FOR C=13 TO 8 STEP
-1: BRIGHT (C=8)
1640 PRINT AT R,C;"S": PRINT AT
R+1,C;"s": BEEP 0.01,-15: PAUSE
5
1650 PRINT AT R,C;"R": PRINT AT
R+1,C;"r": PAUSE 3
1660 PRINT AT R,C;" ": PRINT AT
R+1,C;" ": NEXT C
1669 REM Imprime la figura en la
```

```
 columna 7 lista para recargar m
unicion
1670 PRINT AT R,C;"R": PRINT AT
R+1,C;"r"
1679 REM Recarga municion con ef
ectos de sonido
1680 FOR G=6 TO 5 STEP -1: FOR H
=18 TO 17 STEP -1: FOR N=2 TO 9
1690 PRINT AT H,G; INK 3;N$(N):
BEEP 0.02,H+G-N
1700 NEXT N: NEXT H: NEXT G
1709 REM Quita la figura de la c
olumna 7
1710 PRINT AT R,C;" ": PRINT AT
R+1,C;" "
1719 REM Reinicializa la impresi
on de municion
1720 LET N=8: LET OUT=0: LET H=1
7: LET G=6
1729 REM Camina hacia el arbol
1730 FOR C=8 TO 13: BRIGHT (C=8)
1740 PRINT AT R,C;")": PRINT AT
R+1,C;"+": BEEP 0.01,-15: PAUSE
5
1750 PRINT AT R,C;"(": PRINT AT
R+1,C;"*": PAUSE 3
1760 PRINT AT R,C;" ": PRINT AT
R+1,C;" ": NEXT C
1769 REM Sobreimpresiona la figu
ra en los arboles
1770 OVER 1: PRINT AT R,14;")":
PRINT AT R+1,14;"+": BEEP 0.01,-
15: PAUSE 5: PRINT AT R,14;")":
PRINT AT R+1,14;"+"
1780 PRINT AT R,14;"(": PRINT AT
 R+1,14;"*": PAUSE 3: PRINT AT R
,14;"(": PRINT AT R+1,14;"*"
1789 REM Sobreimpresiona las pie
rnas en el arbol
1790 PRINT AT R+1,15;"+": BEEP 0
.01,-15: PAUSE 5: PRINT AT R+1,1
5;"+"
1800 PRINT AT R+1,15;"*": PAUSE
3: PRINT AT R+1,15;"*"
1809 REM Hace ruidos
```

```
1810 FOR J=1 TO 2: BEEP 0.01,-15
: PAUSE 11: NEXT J
1819 REM Sobreimpresiona al sali
r la figura del arbol
1820 PRINT AT R,18;")": PRINT AT
 R+1,18;"+": BEEP 0.01,-15: PAUS
E 5: PRINT AT R,18;")": PRINT AT
 R+1,18;"+"
1830 PRINT AT R,18;"(": PRINT AT
 R+1,18;"*": PAUSE 3: PRINT AT R
,18;"(": PRINT AT R+1,18;"*"
1839 REM Camina del arbol a la t
ienda
1840 OVER 0: FOR C=19 TO 24: PRI
NT AT R,C;")": PRINT AT R+1,C;"+
": BEEP 0.01,-15: PAUSE 5
1850 PRINT AT R,C;"(": PRINT AT
R+1,C;"*": PAUSE 3
1860 PRINT AT R,C;" ": PRINT AT
R+1,C;" ": NEXT C
1869 REM Reinicializa las variab
les de fila y columna para usarl
as con la rutina del avion
1870 LET C=0: LET R=2: BRIGHT 1
1880 RETURN

1900 REM municion/misil
1902 REM Datos de salida OUT
1909 REM Quita una unidad de la
municion y comprueba si se ha ac
abado
1910 PRINT AT H,G;N$(N): LET N=N
-1
1920 IF N=0 THEN  LET N=8: LET H
=H+1: IF H=19 THEN  LET H=17: LE
T G=G-1: IF G=4 THEN  LET OUT=1
1930 RETURN

2000 REM direccion del misil
2009 REM Borra el misil antiguo
por sobreimpresion y calcula la
nueva posicion
2010 PRINT AT Y,X;M$: LET Y=Y-3:
 LET X=X-2: GO TO 2100
2020 PRINT AT Y,X;M$: LET Y=Y-3:
```

```
 LET X=X-1: GO TO 2100
2030 PRINT AT Y,X;M$: LET Y=Y-3:
 GO TO 2100
2040 PRINT AT Y,X;M$: LET Y=Y-3:
 LET X=X+1: GO TO 2100
2050 PRINT AT Y,X;M$: LET Y=Y-3:
 LET X=X+2: GO TO 2100
2060 PRINT AT Y,X;M$: LET Y=Y-2:
 LET X=X+2: GO TO 2100
2070 PRINT AT Y,X;M$: LET Y=Y-2:
 LET X=X+3: GO TO 2100
2098 REM Si el misil esta en la
misma columna que el avion compr
ueba la fila
2099 REM Si el misil esta sufici
entemente cerca explota y permit
e otro disparo
2100 IF X=C THEN  IF ABS (Y-R)<2
 THEN  GO SUB explode: LET m=mis
sile: GO TO m

2109 REM Si el misil esta fuera
de pantalla permite otro disparo
2110 IF X<0 OR Y<0 THEN  LET m=m
issile: GO TO m

2119 REM Imprime el misil en la
nueva posicion
2120 PRINT AT Y,X;M$: RETURN

2200 REM misil/listo para tirar
2210 LET X=3: LET Y=16
2220 RETURN

2500 REM explosion
2508 REM El misil alcanza el avi
on y suma puntos
2509 REM Comprueba si la explosi
on ha sido en la nube o en el ci
elo
2510 OVER 0: LET SKY=(R<>3 OR C<
16): IF NOT SKY THEN  PRINT AT R
,C-1;"     "
2519 REM Parpadeo y ruido, suma
puntos
```

```
2520 INK 2: IF SKY THEN  PRINT A
T R,C;"t"
2530 FOR I=1 TO 5: BEEP 0.01,I-1
0: BEEP 0.01,-I-10: NEXT I: LET
SC=SC+1
2539 REM Si la explosion es visi
ble, imprime una nube de restos
2540 IF SKY THEN  PRINT AT R-1,C
-1;">@<": PRINT AT R,C-1;"$!&":
PRINT AT R+1,C-1;":%;"
2549 REM Ruido agudo
2550 FOR I=-4 TO 4: BEEP 0.002,(
50+ABS I): NEXT I
2559 REM Borra la explosion
2560 INK 0: IF SKY THEN  PRINT A
T R-1,C-1;"   ": PRINT AT R,C-1;
"   ": PRINT AT R+1,C-1;"   "
2569 REM Imprime la linea de pun
tuacion, reestablece los puntero
s del avion y el misil
2570 INK 8: GO SUB status: LET m
=missile: LET p=plane
2580 LET R=2: LET C=0
2590 RETURN

3000 REM bombardeo del antiaereo
3009 REM Si el campamento no est
a destruido o no quedan bombas n
o se lanza
3010 IF NOT HIT OR NOT BOMB  THE
N  RETURN
3019 REM Pone una bomba en la pa
ntalla e inicia el puntero en la
 cascada
3020 PRINT AT 13,5;".": LET bomb
gun=3030: RETURN
3030 PRINT AT 13,5;" ": PRINT AT
 15,4;".": LET bombgun=3040: RET
URN
3040 PRINT AT 15,4;" ": PRINT AT
 17,3;".": LET bombgun=3050: RET
URN
3050 PRINT AT 17,3; FLASH 1;" ":
 LET bombgun=3060: RETURN
3059 REM Quita el avion de panta
```

```
lla; comprueba si quedan misiles
 en pantalla para eliminarlos
3060 PRINT AT 12,0;"      ": OVER
 1: FOR I=1 TO 5: GO SUB m: NEXT
 I
3069 REM Parpadea el borde y hac
e ruidos de explosion
3070 FOR I=1 TO 50: BORDER RND*7
: BEEP 0.01,RND*30-15: NEXT I
3079 REM Fija la tinta a azul y
hace el avion invisible
3080 LET DEAD=1: OVER 0: INK 5
3089 REM Pone a cero la cascada
3090 LET bombgun=3000: RETURN

3500 REM bomba en campamento
3509 REM Si el campamento esta d
estruido reserva la bomba para e
l tirador
3510 IF HIT  THEN  RETURN
3519 REM Resta una bomba como la
nzada y comienza la cascada
3520 LET BOMB=0: PRINT AT 14,21;
".": LET bombcamp=3530: RETURN
3530 PRINT AT 14,21;" ": PRINT A
T 15,22; BRIGHT 0;".": LET bombc
amp=3540: RETURN
3540 BRIGHT 0: PRINT AT 15,22;"
"
3549 REM Explota la tienda
3550 PRINT AT 16,23; OVER 1; INK
 2;"#$"
3560 PRINT AT 17,23; OVER 1; INK
 2;"%&"
3570 LET bombcamp=3580: BRIGHT 1
: RETURN
3579 REM Borra la explosion
3580 BRIGHT 0: PRINT AT 16,23; O
VER 1; INK 0;"#$"
3590 PRINT AT 17,23; OVER 1; INK
 0;"%&"
3600 LET bombcamp=3610: BRIGHT 1
: RETURN
3609 REM Borra la linea superior
 que queda de la tienda
```

```
3610 PLOT  INVERSE 1;184,33+YT:
DRAW  INVERSE 1;16,0
3619 REM Comprueba que la tienda
 ha sido destruida y pone a cero
 la cascada
3620 LET bombcamp=3500: LET YT=Y
T-1: LET HIT=(YT=0)
3630 RETURN

4600 REM carga
4609 REM Carga la pantalla del c
asete
4610 LET N$="escenario"
4620 INPUT ("  CARGANDO "+N$+CHR
$ 6+"Arranca el casete, da ENTER
."); LINE X$
4630 LOAD (N$)SCREEN$
4640 RETURN

5000 REM crear/caracteres
5009 REM Crea los caracteres par
a la municion en los G.D.U.
5010 LET D=255
5020 FOR I=0 TO 6: FOR J=0 TO 7
5030 LET P=USR "G"-I*8+J: POKE P
,0
5040 IF J>I THEN  POKE P,D
5050 NEXT J: NEXT I
5060 RETURN

5500 REM puntuacion
5509 REM Imprime en las dos ulti
mas lineas
5510 DEF FN S(R,C)=16384+INT (R/
8)*2048+(R-INT (R/8)*8)*32+C
5520 DEF FN C(A$)=15360+CODE A$*
8
5530 LET R=22: LET A$="DEFENSA D
E LA ISLA POR BJJ & IOA"
5540 FOR J=1 TO LEN A$: LET C=J-
1: GO SUB char: NEXT J
5550 LET R=23: LET A$="MAX.PUNTO
S       POR BJJ  "
5560 FOR J=1 TO LEN A$: LET C=J-
1: GO SUB char: NEXT J
```

```
5570 RETURN

5600 REM caracter
5609 REM Imprime el caracter jes
imo de A$ en R,C, cambia el atri
buto a papel rojo, tinta blanca
5610 LET AT=FN S(R,C): LET FROM=
FN C(A$(J))
5620 FOR I=0 TO 7: POKE AT+I*256
,PEEK (FROM+I): NEXT I
5630 POKE 22528+C+R*32,87: BEEP
0.03,CODE A$(J)-50
5640 RETURN

5700 REM maxpuntos
5709 REM Cambia la maxima puntua
cion y la imprime en la parte ba
ja
5710 LET HSC=SC: LET A$=STR$ HSC
5720 LET R=23: FOR J=1 TO LEN A$
: LET C=10+J: GO SUB char: NEXT
J
5729 REM Cambia el nombre a letr
as parpadeantes
5730 FOR I=23291 TO 23293: POKE
I,215: NEXT I
5739 REM Lee 3 iniciales del tec
lado para la maxima puntuacion
5740 LET J=1: FOR C=27 TO 29
5750 IF INKEY$<>"" THEN  GO TO 5
750
5760 IF INKEY$="" THEN  GO TO 57
60
5770 LET A$=INKEY$: GO SUB char:
 NEXT C
5780 RETURN
```

Caracteres del juego

Es fundamental planificar cuidadosamente el juego antes de comenzar a escribir el programa. Se debe empezar por diseñar un plan general de la escena del juego propuesto en un papel de gráficas. A continuación colocaremos los objetos fijos en sus posiciones (en nuestro caso el sol y las nubes) así como las áreas que serán posteriormente atravesadas por objetos en movimiento (por ejemplo, el pasillo de

bloques que en algún momento contendrán los caracteres de dibujo de los aviones). Con ello conseguiremos fijar la escala de los objetos a utilizar, y obtendremos una idea del aspecto final del juego acabado. Téngase presente que el tiempo consumido en una buena planificación en ese momento representa una mínima fracción del que se emplearía para ajustar un programa a punto de terminarse. Se deberá asegurar de que la pantalla a dibujar puede realmente ajustarse dentro del área de gráficos, y de que nunca se llegará a una situación en la que aparezcan más de dos colores simultáneamente en el mismo bloque. Una vez decidido el fondo de la escena, nos dedicaremos a crear los objetos que aparecerán en primer plano (por ejemplo, explosiones, el avión, el hombre), es decir, las partes en movimiento. Para colocar un objeto grande en la pantalla el comando BASIC más rápido es la sentencia PRINT. Construiremos, por tanto, nuestros objetos en forma de bloques de caracteres definidos, y utilizaremos el programa de generación de caracteres del capítulo 5 para producir las formas requeridas. Deberemos examinar repetidamente los caracteres a su tamaño normal y editarlos cuantas veces sea necesario hasta conseguir exactamente la forma deseada. Puede resultar de utilidad añadir partes de su programa de juego al final del programa de generación de caracteres (opción 7; véase capítulo 5). De esta forma podremos observar los objetos en acción sin haber tomado una decisión final sobre su forma.

Ejercicio 14.1

Añada una nueva opción al programa GENERACION DE CARACTERES de manera que se pueda editar un único carácter de entre un grupo. La opción deberá introducir (POKE) los ocho valores BINarios del carácter que está editándose, junto con los demás caracteres del grupo, en las posiciones de *display file* pertinentes de la pantalla; por ejemplo, los dos bloques que forman el avión en vuelo horizontal hacia la derecha (véase figura 14.1). Con ello podemos decidir rápidamente las ventajas e inconvenientes de alterar algún *pixel* del carácter.

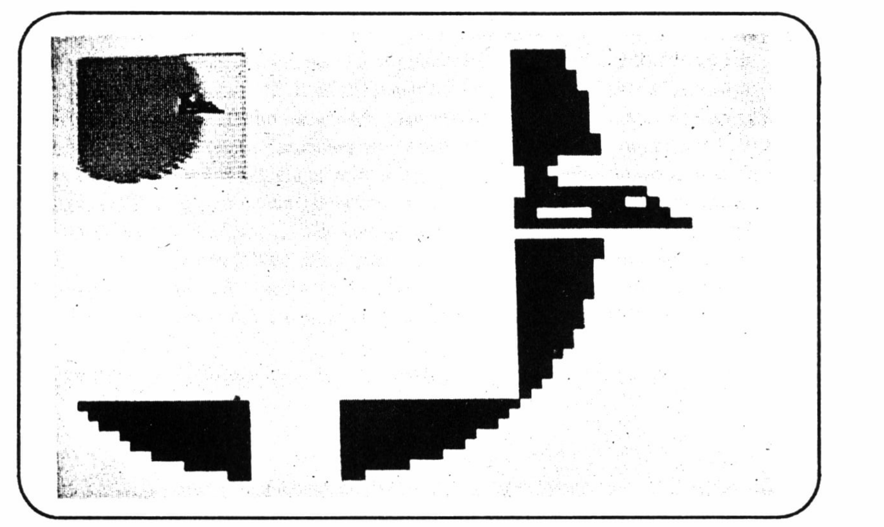


Figura 14.1

Fondo de la escena

A continuación se crea el fondo utilizando una combinación del GENERADOR DE CARACTER y las subrutinas de construcción de diagramas del capítulo 6. Como primera medida rellenamos, utilizando "papel", áreas rectangulares de bloques que cubran aproximadamente las partes de la escena que se desee (por ejemplo, hierba, cielo, sol, etc.). A continuación sobreimprimimos los detalles iniciales por medio de caracteres especiales y órdenes PRINT (por ejemplo, los árboles, sacos de arena, "punto" y "línea" de las subrutinas de diagramas. Por ejemplo, los árboles se realizan añadiendo líneas y puntos a los caracteres utilizados para la explosión. Podemos aumentar el número de opciones incluyendo colores con BRIGHT junto con colores normales. Debe tenerse un cuidado especial en la construcción de aquellos bloques que serán ocupados en algún momento por objetos en movimiento. Dichos bloques deberán contener un solo color de fondo sin ningún detalle adicional. Recuérdese que sólo se permite tener dos colores simultáneamente en el mismo bloque, por lo que existe siempre el peligro de que un objeto en movimiento introduzca un tercer color. En nuestro caso particular (véase figura 14.2), existen dos zonas del fondo en las que deberemos proceder con precaución.

1) Los aviones pasan por delante y por detrás de una nube en su camino de vuelta, de manera que la nube debe contener una "puerta" de bloques blancos en su centro. Obsérvese que no podemos colocar el cielo azul y la nube blanca en el mismo bloque por el que ha de pasar el avión, de manera que la frontera entre cielo y nubes en esta puerta debe ser un borde recto. El avión desaparece realmente detrás de la nube durante su vuelo, de tal forma que algunos de los bloques de esta puerta contienen tinta (INK) blanca y papel (PAPER) blanco. Así pues,

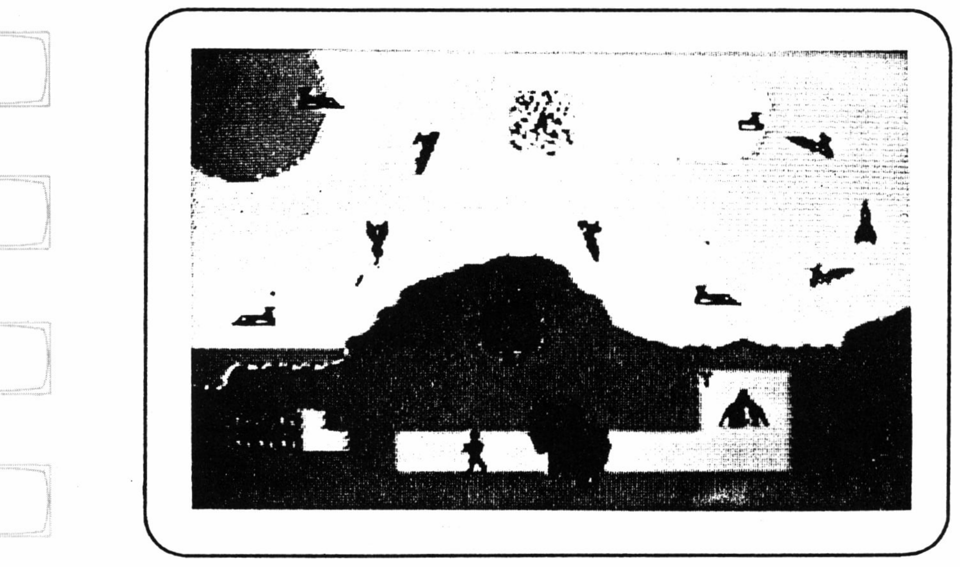


Figura 14.2

cuando el avión penetra en este bloque, los *pixels* de tinta toman el mismo color blanco del papel, y el avión resulta indistinguible respecto a la nube.

2) El sol, que parece ser un círculo suave, tiene en realidad bordes planos en cada parte. Los aviones pasan a través de uno de estos bordes, sin encontrar en ningún momento una combinación de sol y cielo en el mismo bloque. Se puede comprobar este último punto en la ampliación de la figura 14.1, la cual ha sido realizada con la ayuda de la subrutina "*pixels* gigantes" del listado 5.2. Este diagrama ignora los atributos verdaderos de los bloques; en realidad, utiliza tinta negra y papel blanco. Los colores verdaderos pueden observarse en el dibujo a escala 1/16 situados en la esquina superior izquierda. Evidentemente, aquellos bloques de caracteres de la escena que no vayan a ser alterados durante el juego pueden contener dos colores.

Animación en cascada

Una vez que hemos almacenado esta primera aproximación del fondo en la cinta, deberemos escribir las subrutinas que posicionarán y moverán los objetos en pantalla. Nuestro principal objetivo será la velocidad de ejecución, por lo que deberemos realizar las subrutinas tan explícitamente como sea posible. Así pues, es necesario minimizar el número de cálculos que se requieran cuando el juego está en funcionamiento. Si se puede realizar una programación extra o una serie de cálculos con anterioridad al comienzo del juego, conseguiremos con ello salvar algunos preciosos milisegundos durante el mismo; cualquier ahorro de tiempo de este tipo es aconsejable aunque la programación previa resulte molesta. Describiremos una técnica de este tipo en nuestro programa ejemplo, la conocida técnica de la *cascada* o método de *punto de entrada variable*. Aquellas subrutinas que sean importantes desde el punto de vista de tiempo de ejecución serán llamadas por medio de un puntero variable (un "*pointer*", identificador que utilizaremos en letras minúsculas). Estas subrutinas componen una cadena (o cascada) de subtareas programadas, de las cuales se emprende cada vez una y sólo una. Una vez realizada la subtarea, el puntero a estas subrutinas se altera y se devuelve el control al programa principal. Al entrar de nuevo, el programa obedece naturalmente a una sección diferente del código dentro de la subrutina; en general, el puntero se dirige a la siguiente subtarea de la cascada. Continuando el proceso, el puntero va moviéndose por las secciones de la cascada hasta alcanzar el fondo de la misma (en donde usualmente se vuelve a inicializar al principio). El juego completo consta de una fase de inicialización seguida por un bucle de llamadas a subrutinas, cada una de las cuales es una cascada que resuelve un problema específico dentro del juego (por ejemplo, mover el avión o bombardear la base). Las subrutinas de cascada pueden incluso llamar a nuevas cascadas. Esta técnica consigue una impresión de proceso paralelo, que es esencial para dar veracidad al juego, de manera que el usuario crea que está participando en un combate en el que pueden suceder varias cosas simultáneamente. En nuestro caso, intentamos dar la impresión de que el avión, el antiaéreo y los misiles pueden moverse independientemente.

De todas formas, este juego es quizá demasiado complicado para dar una idea elemental de la técnica; por ello nos referiremos a otro, mucho más simple, que también puede ejecutarse en máquinas de 16K. Observemos los dos programas siguientes que ejecutan funciones independientes: el listado 14.2 espera hasta que se pulsa una tecla, y a continuación dispara un punto por la pantalla; el listado 14.3, por su parte, mueve un signo más por toda la pantalla en zig-zag. Ambos programas utilizan las técnicas de animación rápida que se pueden encontrar en el programa DEFENSA DE LA ISLA; sin embargo, debido a su pequeño tamaño, es necesario retardarlos con un comando PAUSE (el exceso de velocidad es problema únicamente en programas muy sencillos). Con estos listados hemos creado dos subrutinas de cascada (listado 14.4) de modo que el punto y la cruz parecen moverse simultáneamente.

La cascada del punto consta de tres partes: *a*) colocar el punto en la fila 11, columna 0; *b*) si el usuario toca el teclado, el punto empieza a moverse; y *c*) mueve el punto columna a columna por la pantalla hasta que alcanza el borde derecho. Una vez finalizado cada uno de estos procesos, el identificador de la subrutina "punto" se mueve a la siguiente sección. Cuando el punto ha alcanzado la parte derecha, "punto" se reinicializa a posición original.

La cascada de la cruz tiene también tres partes: *a*) colocar la cruz en la fila 22, columna 8; *b*) mover la cruz una fila arriba y en diagonal hacia la derecha en la mitad inferior de la pantalla; y *c*) mover la cruz una fila arriba y en diagonal hacia la izquierda en la mitad superior; además, si el punto y la cruz coinciden en el mismo bloque, chocan.

Las dos subrutinas se han transformado en un juego con un programa principal que llama repetidamente a "*dot*" y "*cross*", las subrutinas "punto" y "cruz", cambiando los identificadores correspondientes de forma constante, de tal manera que se pueda acceder en cada momento a la parte correcta de la cascada.

Listado 14.2

```
 200 REM punto
 210 PRINT AT 11,0;"."
 220 IF INKEY$ = "" THEN  GO TO
220
 230 LET D = 0
 240 PRINT AT 11,D;" ": LET D =
D + 1
 250 IF D = 32 THEN  GO TO 210
 260 PRINT AT 11,D;"."
 270 PAUSE 1: GO TO 240
```

Listado 14.3

```
300 REM cruz
310 LET R = 21: LET C = 8
320 PRINT AT R,C;"+"
330 PRINT AT R,C;" ": LET R = R
- 1
340 IF R < 0 THEN  GO TO 310
350 IF R > 11 THEN  LET C = C +
1
360 IF R <= 11 THEN  LET C = C
- 1
370 PRINT AT R,C; "+"
380 PAUSE 1: GO TO 330
```

Listado 14.4

```
 100 REM bucle principal
 110 LET D=0: LET dot=200: LET c
ross=300
 120 GO SUB dot: GO SUB cross
 130 GO TO 120

 200 REM cascada de punto
 210 PRINT AT 11,0;".": LET dot=

220: RETURN
 220 IF INKEY$="" THEN  RETURN
 230 LET D=0: LET dot=240: RETUR
N

 240 PRINT AT 11,D;" ": LET D=D+
1
 250 IF D=32 THEN  LET dot=210:
RETURN
 260 PRINT AT 11,D;"."
 270 RETURN

 300 REM cascada de cruz
 310 LET R=21: LET C=8
 320 PRINT AT R,C;"+": LET cross
```

```
=330: RETURN
 330 PRINT AT R,C;" ": LET R=R-1
 340 LET C=C+1
 350 PRINT AT R,C;"+"
 360 IF R>11 THEN  RETURN
 370 LET cross=380: RETURN

 380 IF R=11 AND D=C THEN  PRINT
 AT R,C;"CRASH": STOP
 390 PRINT AT R,C;" ": LET R=R-1
 400 IF R<0 THEN  LET cross=310:
 RETURN
 410 LET C=C-1
 420 PRINT AT R,C;"+"
 430 RETURN
```

Ejercicio 14.2

Añada una línea al bucle de llamada del programa anterior de manera que, tras un choque, los punteros se reinicialicen al comienzo de las cascadas y se imprima el número de aciertos.

Escriba un juego de "tiro al blanco" en que un cazador, controlado por teclado, se mueva a izquierda y derecha en la parte inferior de la pantalla. Debe disparar a unos patos que vuelan de izquierda a derecha, subiendo y bajando, por la misma.

Técnicas de animación adicionales

En el ejemplo anterior, el ahorro de tiempo y esfuerzo de programación conseguido por las técnicas de cascada es realmente pequeño. Sin embargo, en programas grandes, en los que aparecen cascadas complicadas (por ejemplo, el movimiento del avión en DEFENSA DE LA ISLA), se pueden conseguir ahorros de tiempo que marque la diferencia entre un atractivo juego, repleto de dinamismo, y un juego lento y aburrido. Obsérvese que hay ocho juegos de caracteres diferentes para describir el avión; véase figura 14.2. En la figura se ve la escena completa y un ejemplo de cada avión así como algunos otros objetos con posibilidad de movimiento. La cascada de dibujo del avión está dividida en secciones, cada una de las cuales sitúa un tipo determinado de avión en un área concreta de la pantalla.

Hemos utilizado en el juego una gran variedad de modos diferentes de resolver problemas de animación. Los aviones se mueven imprimiéndolos con tinta transparente y eliminando su posición anterior con blanco. Por su parte, los misiles se manejan por medio de órdenes OVER (también con tinta transparente) eliminándolos de su posición por medio de un OVER del mismo carácter. Ambos métodos presentan ventajas y desventajas. Cuando se pueda combinar la eliminación de.la posición anterior con la creación de la nueva (véanse las secciones en que el avión vuela

horizontalmente) observará que una orden PRINT lleva en general menos tiempo, pero por contra se pierde cualquier detalle que hubiera en el fondo. La orden OVER, por el contrario, es más lenta, pero deja el fondo intacto. Si utilizamos una combinación de estas técnicas en un programa, podemos encontrarnos con problemas cuando dos objetos en movimiento pasan simultáneamente por el mismo bloque: por ejemplo, ¿qué sucedería si el cuadrado que contiene un misil queda tapado por un avión que pasa por el mismo sitio? Esta circunstancia se puede dar en nuestro juego en el bloque de la fila 7, columna 9. Lo que sucede es que el misil que ha sido sobreimpresionado, lo que supuestamente ha cancelado el antiguo, queda colgando en medio del aire. En estos casos (lo cortés no quita lo valiente) es mucho mejor tratar el problema explícitamente y "filtrarlo" en el programa, en lugar de complicar el algoritmo general, introduciendo probablemente algún otro error en el mismo. En la cascada del avión se ha incluido una sentencia extra (línea 1190), con la que nos aseguramos de que se cubre rápidamente cualquier "olvido". Cuando trate de eliminar estos defectos, recuerde siempre que una instrucción que oculte el objeto "por las bravas" será probablemente bastante más veloz (tanto en tiempo de ejecución como en tiempo de programación) que una maravillosa subrutina sin ningún fallo. Aunque hayamos repasado cuidadosamente las peculiaridades de nuestro juego en la etapa de planificación, es bastante probable que aparezcan algunos defectos de este tipo.

En nuestro programa hay dos lugares en los que mostramos un objeto pasando aparentemente por detrás de otro objeto perteneciente al fondo. Para conseguir que el avión desaparezca detrás de la nube, simplemente hacemos parte de sus bloques con tinta blanca, con el efecto subsiguiente de que el avión desaparece. En realidad, nos aprovechamos de que un avión blanco sobre una nube blanca es tan "fácil" de ver como un gato negro en una noche sin luna, por lo que aparentemente el avión se ha ocultado en la nube. El otro caso aludido se refiere al hombre que camina por el sendero, pasando por detrás del árbol: esta vez el problema es bastante más complicado. Necesitamos que el árbol continúe teniendo dos colores cuando el hombre está detrás de él. La estratagema utilizada en esta ocasión es hacer que el hombre camine hasta el árbol normalmente; sobreimpresionarlo (OVER) cuando coincide con el mismo bloque que el follaje exterior, y reducir exclusivamente a ruidos de marcha, sin imprimir nada durante una cantidad de tiempo tal que parezca que el hombre ha pasado por detrás del árbol y ha aparecido por la otra parte. Por otra parte, las piernas deberán aparecer por detrás del árbol un bloque antes que la cabeza cuando se está moviendo hacia la derecha, y realizar un esquema similar cuando se mueve hacia la izquierda. Todos estos detalles se deben calcular cuidadosamente antes de escribir la primera línea del código BASIC.

Con una combinación de estas técnicas de movimiento de objetos, y con posibilidad de pasar de una a otra, llegaremos a construir pantallas de muy alta calidad. Si desea juegos realmente rápidos y complejos, por supuesto, deberá utilizar subrutinas de código máquina. Aun así, muchas de las subrutinas podrán todavía estar escritas en BASIC. No hay ninguna razón para escribir el programa completo en código máquina a no ser que usted pretenda vender sus juegos.

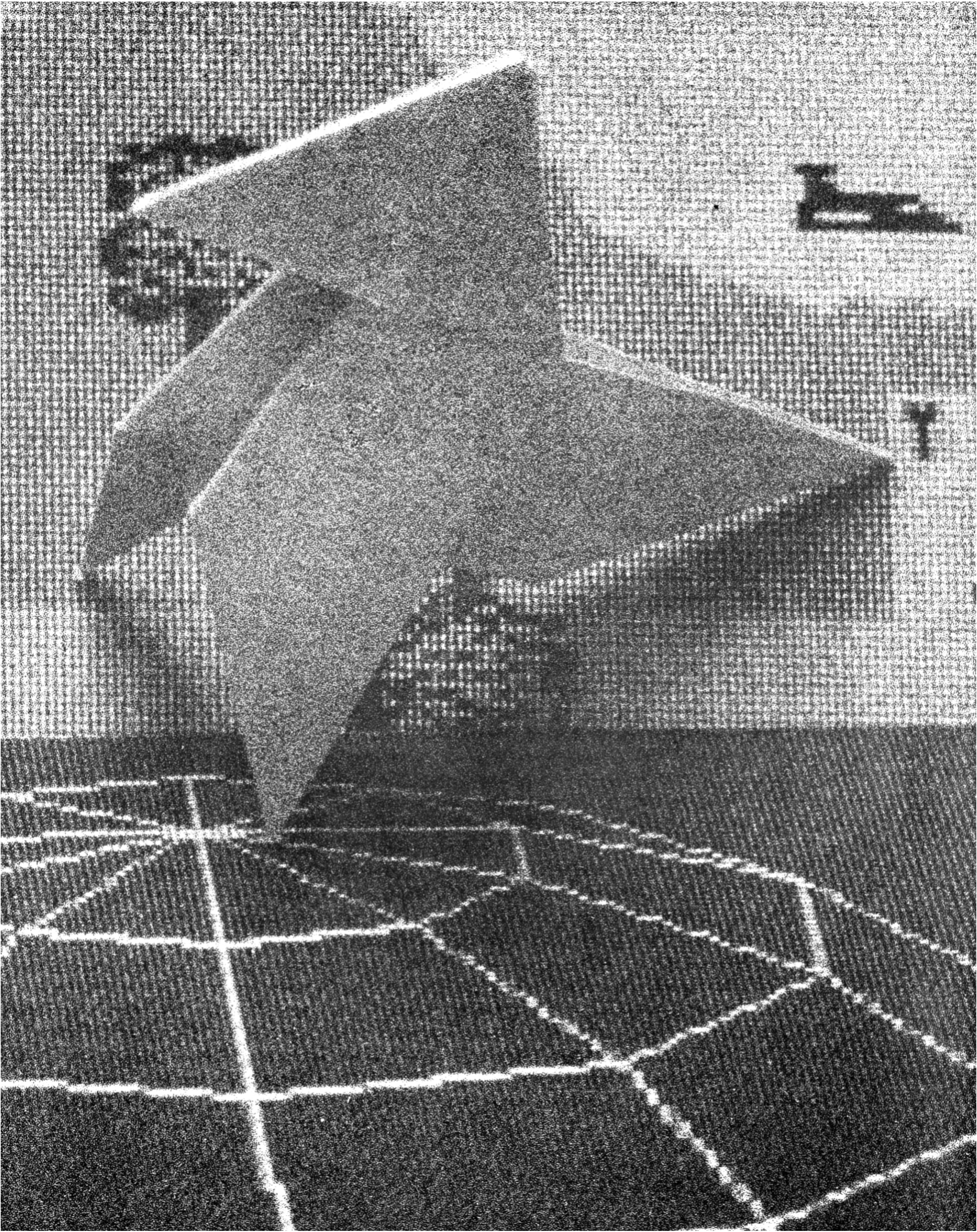
Un consejo final. Conforme sus programas de juego se vayan haciendo más y más complicados e interconectados, procure mantener siempre una visión global del programa. Tenga claro el trabajo que se debe realizar en el nivel más alto. Utilice nom-

bres de variables que tengan un significado fácilmente comprensible; llene el programa de comentarios durante el desarrollo, y, sobre todo, ¡que no cunda el pánico!

Le dejamos ahora con su Spectrum. Ha demostrado ser una herramienta eficaz y robusta, directa y fácil de utilizar. Estamos seguros de que tiene por delante muchas horas (años quizá) de diversión con esta máquina. Para comenzar, le daremos una serie de ideas en el siguiente capítulo. Buena suerte y buenos programas.

Programas completos

I. Listado 14.1: DEFENSA DE LA ISLA. Cargue el programa directamente de la cinta, ya que teclearlo le llevará un tiempo excesivo, no sólo por el listado, sino por los juegos de caracteres que lo acompañan. Si lo desea, produzca sus propios juegos de caracteres y fondo.
II. Listado 14.2, "punto" *(dot)*. Pulse cualquier tecla.
III. Listado 14.3, "cruz" *(cross)*. No necesita datos.
IV. Listado 14.4, "programa principal", "cascada de punto" y "cascada de cruz". Pulse cualquier tecla.



15
Proyectos

I. Utilice su Spectrum para dibujar un reloj digital. Use caracteres especiales de gran tamaño para los dígitos, separando las horas de los minutos con un símbolo dos puntos. Consiga que el reloj marque la hora exacta utilizando el reloj interno del Spectrum (véase Manual BASIC de Spectrum).

II. Escriba un programa que compruebe los conocimientos de código Morse del usuario. El programa deberá contener un párrafo de texto; una vez traducido a Morse, el Spectrum deberá imprimir los puntos y rayas correspondientes utilizando bloques de caracteres de baja resolución. Utilice también la función BEEP para simular el sonido del Morse. El programa deberá llevar también una variable en la que se especifique la velocidad de producción de código Morse, de tal manera que el test se ejecute más rápidamente conforme el usuario va adquiriendo experiencia.

III. Dibuje un conjunto de señales de tráfico internacionales. El programa deberá dibujar el fondo de la figura (por ejemplo, un triángulo rojo); utilice a continuación subrutinas especiales escritas por usted mismo, o los programas de los capítulos 5 y 6 para dibujar los motivos centrales.

IV. Realice crucigramas en su televisor. Cada recuadro del crucigrama deberá tener un tamaño de 2 por 2 bloques. Los cuatro bloques podrán ser negros (en cuyo caso no se introducirá ninguna letra) o blancos, en los que la esquina inferior izquierda contendrá la letra de la solución, y los dos caracteres superiores el número de fila o columna correspondiente (si lo tiene). Dispone de espacio suficiente para un crucigrama de 16 por 11.

Necesitará también poner el texto con las indicaciones de horizontales y verticales en la pantalla, por lo que se encontrará falto de espacio. Este problema se puede resolver utilizando las ideas de "proyección de diapositivas" del capítulo 13; coloque el crucigrama en uno de los encuadres y la clave en los cuatro encuadres restantes. Las soluciones al crucigrama se pueden añadir por medio de un programa "cursor", o cualquier otro código de entrada especial; por ejemplo, la letra "H" (horizontales) o "V" (verticales) seguidas por el número correspondiente, y a continuación la solución. Si realiza un crucigrama de menor tamaño, podrá también escribir las soluciones en la propia pantalla, de una en una (o quizá colocándolas en las filas 23 y 24), en cuyo caso no necesita retirar el crucigrama de la pantalla.

V. En el Manual de Spectrum se da un programa para dibujar una bandera. Escriba programas o utilice el generador de caracteres y las subrutinas de diagramas (capítulos 5 y 6) para dibujar otras. También puede dibujar escudos de compañías o diseñar motivos nuevos. Utilice las técnicas del capítulo 13 para acceder a las posiciones de *display file* con el fin de añadirlo como opción extra a las subrutinas de diagramas. Esta opción le permitirá copiar un conjunto de bloques (que ya estén en pantalla especificados por "cursor") sobre otro conjunto del mismo tamaño que se encuentre en otro lugar de la pantalla (también especificado por "cursor"). También puede girar o reflejar el motivo realizado.

VI. En el Manual de Spectrum se muestra cómo se puede utilizar la función BEEP para producir música (?). Escriba un programa que dibuje en pantalla la partitura mientras BEEP interpreta los sonidos. Dibuje los pentagramas y utilice caracteres especiales para las notas, pausas, etc. Incorpore un metrónomo para marcar el ritmo de la canción.

VII. Use el método de los bloques de caracteres para dibujar laberintos. El programa, obviamente, deberá generar laberintos con soluciones reales. Imponga limitaciones de tiempo para la búsqueda de la salida. Haga laberintos animados, que cambien conforme avanza el juego. Añada alicientes: monstruos comehombres que pululan por el laberinto; trampas que aparecen de repente y pueden tragarles; "agujeros negros" que lo trasfieran a otro lugar del laberinto si no se mueve lo suficientemente rápido.

VIII. Realice extensiones de las ideas del capítulo 6. Dibuje sus propios histogramas especiales, círculos porcentuales y gráficas. Introduzca animación en los diagramas (con el método "cine" del capítulo 13 o con el "juego del gusano" del capítulo 1, y su forma extendida del capítulo 14). Genere juegos completos de caracteres especiales. Fabrique histogramas (aparentemente) en tres dimensiones dibujando cada barra con tres secciones de diferente color. La sección frontal de cada barra deberá ser un número entero de bloques de caracteres, y tendrá una anchura de dos bloques. La sección lateral será de un bloque de ancho, de la misma altura que la anterior, terminando en su parte inferior en un triángulo que se una a la sección frontal. La parte superior estará colocada sobre la sección frontal, y tendrá forma de rombo, uniendo las dos anteriores.

De esta forma se asegura que no habrá en ningún caso más de dos colores en el mismo bloque, y da a la barra la apariencia de una perspectiva ortogonal de una columna rectangular.

IX. Produzca composiciones abstractas. Utilice OVER con un gran número de líneas aleatorias sobre la pantalla. Alternativamente, puede dibujar líneas siguiendo pautas regulares muy densas, con lo que obtendrá diagramas de Moire. Por ejemplo, dibuje las líneas que unen los puntos (0, I) a (255, 175 − I) para $0 \leqslant I \leqslant 175$ y los puntos (I, 0) a (255 − I, 175) para $0 \leqslant I \leqslant 255$. Extienda las ideas del programa de composiciones del capítulo 5 para producir composiciones simétricas complejas: cualquier libro de introducción a la cristalografía les aportará un gran número de ideas.

X. Los libros de cristalografía le proporcionarán ideas sobre objetos tridimensionales. Extienda los programas a cuatro dimensiones: los vértices serán ahora un vector de cuatro números y las transformaciones necesitarán matrices 5×5. Una proyección ortogonal en el espacio de cuatro dimensiones se realizará ignorando dos de las coordenadas (en lugar de una, *z*, como se hacía en tres dimensiones). ¿Cómo se realizan traslaciones, escalas y rotaciones en cuatro dimensiones?

XI. Hasta ahora hemos presentado dos juegos en tablero, ajedrez y *Master Mind*. Hay muchas otras posibilidades: damas, *Scrabble*, el ahorcado, parchís. Puede crear un fichero completo de juegos de este tipo. El Spectrum puede utilizarse simplemente como tablero o actuar también de árbitro. Si se siente realmente emprendedor, puede incluso intentar utilizar el ordenador como oponente.

XII. Utilice caracteres especiales para construir una baraja de cartas. Puede incorporarlas a un programa que juegue *blackjack* actuando el Spectrum como banquero y oponente.

XIII. Puede dibujar algunos tipos de rompecabezas en su televisor. Por ejemplo, supongamos que tenemos nueve cuadrados, cada uno de ellos dividido en cuatro partes por sus diagonales. Cada parte tiene un color (azul "1", rojo "2", magenta "3" y verde "4") y un signo ("+" o "−"). Representaremos cada cuadrado como una secuencia de cuatro números que denote las áreas de cada color leídas en sentido horario respecto al centro. Por ejemplo, podemos tener (−1, −2, 1, 4), (−1, 3, 4, −2), (1, −4, −2, 3), (1, 2, −3, −4), (1, 3, −2, −4), (1, 4, −3, −2), (2, −3, −4, 3) y dos casillas (−1, −4, 3, 2). El problema a resolver consiste en colocar los nueve cuadrados en una dis-disposición tres por tres, de manera que cada dos partes vecinas que se toquen sean del mismo color pero de signo opuesto. Puede utilizar el Spectrum para dibujar los cuadrados inicialmente en la parte izquierda de la pantalla y construir un retículo tres por tres del mismo tamaño en la derecha. A continuación tomará los cuadrados de la izquierda y los colocará en el retículo, o los volverá a dejar en su posición original a la izquierda.

Además, puede escribir un programa que encuentre la solución del problema anterior; su ejecución lleva unos diez minutos, y encuentra dos soluciones independientes.

XIV. Produzca un paquete de gráficos de resolución media para manipular bloques 2×2. Este paquete será similar al que apuntamos en el capítulo 5 para bloques de caracteres. La pantalla consistirá, por consiguiente, en un conjunto de 64 por 44 bloques. Tómese una fotografía de sí mismo y superpóngala a un retículo de 64 por 44. En cada cuadrado, decida si es predominantemente claro u oscuro y coloree el bloque de acuerdo con ello. Aparentemente es un trabajo muy pasado, pero observe que la mayor parte de la figura será de gris claro, por lo que si usa papel blanco y tinta negra, la mayor parte de los cuadrados se podrán despreciar. Como una cabeza es más alta que ancha, podrá conseguir mejor resolución si realiza el dibujo "acostado" en la pantalla. También puede dibujar dos cabezas verticales en la pantalla.

XV. Escriba un videojuego del tipo "comecocos". Necesitará preparar cinco objetos que se muevan por la pantalla al mismo tiempo. Para conseguir que el juego funcione más rápido, haga que sólo dos de los "fantasmas" se muevan por cada movimiento del "comecocos". Los fantasmas deberán ser capaces de encontrar el camino más corto hacia el jugador cuando están en modo ataque, y la mejor ruta de escape cuando se encuentran huyendo. Si encuentra demasiadas dificultades para encontrar el algoritmo correcto, haga que los fantasmas se muevan simplemente hacia (o alejándose de) el jugador, mientras no haya una pared que se lo impida. Busque una forma rápida de codificar estos movimientos, ya que será la parte del juego que más tiempo consuma. Tenga siempre presente que la velocidad es la esencia de un buen videojuego. Si añade algunas pequeñas subrutinas en código máquina para imprimir las cinco figuras en la pantalla, conseguirá mejores resultados; puede moverlas a continuación utilizando órdenes "PRINT AT".

XVI. Escriba un programa que como primera providencia presente (OVER) en menú de símbolos especiales en la parte izquierda de la pantalla, por ejemplo, las figuras utilizadas de componentes de circuitos electrónicos (resistencias, condensadores, etc). Cada uno de estos símbolos estará formado por un conjunto de bloques de caracteres. Utilice un cursor para apuntar a cualquier símbolo del menú y a continuación (utilizando OVER) llevar una copia del mismo a la posición requerida de la pantalla. Incluya además la posibilidad de dibujar líneas de conexión y etiquetar con caracteres numéricos y especiales reducidos (por ejemplo, Ω para ohmios). También deberá tener en cuenta la posibilidad de borrar un símbolo colocado con posición incorrecta. Añada también opciones como almacenar y cargar un diagrama realizado, así como la posibilidad de borrar el menú una vez conseguido el dibujo final.

XVII. En todos nuestros programas de dibujos en perspectiva hemos supuesto que los objetos estaban totalmente situados delante del ojo. Cambie los programas de manera que puedan tratar el caso general en el cual algunos vértices puedan estar situados detrás del mismo.

Apéndice A
Adaptación de programas al Spectrum 16K

En este modelo de ordenador, algo más de 7K se utilizan en la pantalla y fichero de atributos, así como para ciertas variables del sistema. Por tanto, el tamaño de los programas y datos a introducir deberá ser inferior a aproximadamente 8,8K. Muchos de los programas presentados en este libro exceden este valor; la mayor parte de ellos, sin embargo, funcionarán siempre que el lector observe las siguientes normas:

1) Borre todas las instrucciones REM de los listados de la cinta, así como cualquier subrutina no utilizada de los ficheros de librería (por ejemplo, "plot" y posiblemente "escala"); realice esta operación antes de ejecutar MERGE con las demás subrutinas. Siga asimismo las instrucciones dadas en el capítulo 13 para optimización de código de programa. Así, por ejemplo, el programa de dibujo del reactor (listados "rut1", "rut3" y "9.9") pueden encajar casi exactamente en la memoria eliminando todos los REM y las subrutinas "escala" y "plot".

2) Es posible que no consiga almacenar simultáneamente los cuatro juegos de caracteres alternativos (conjuntos 2 al 5) y el juego de Gráficos Definidos por Usuario (conjunto 6). Si consigue espacio suficiente para todos (contando con el conjunto número 1, que es el estándar), la tabla de direcciones correspondientes deberá contener los valores 15360, 29271, 30039, 30807, 31575 y 32080 respectivamente. Si utiliza algún juego con dirección menor, puede llegar a "corromper" su programa y/o datos, o incluso *colgar* el ordenador. ¡Utilice únicamente los conjuntos que no interfieran con el almacenamiento! La sentencia CLEAR situada al comienzo de los programas que usan caracteres alternativos deberá cambiarse de CLEAR 62294 a CLEAR 255 + la dirección del conjunto disponible más bajo (excepto el 1). Se encontrará con que en el programa GENERADOR DE CARAC-

TERES no hay espacio suficiente para el conjunto 2, y en este caso deberá introducir un CLEAR 255 + 30039. Si realiza un programa que necesite el conjunto 2, cargue estos caracteres como conjunto 5 (por ejemplo), guárdelos en cinta, y cárguelos en su programa, el cual, por supuesto, deberá disponer de espacio para dicho conjunto.

3) Los programas se pueden dividir en partes no dependientes entre sí, y almacenar los gráficos y/o datos producidos en cinta. A continuación se cargarán estos ficheros junto con otros programas, para continuar su manipulación; esta técnica, por ejemplo, resultará de utilidad en las construcciones de diagramas del capítulo 6. Deberá cargar (LOAD) "rutdiag" y encadenar (MERGE) uno de los programas de histogramas, círculos porcentuales, gráficas o edición de dibujos. Durante la ejecución del programa deberá tener en cuenta que no debe intentar acceder, obviamente, a las subrutinas que no se encuentren en memoria en ese momento. Podrá guardar (SAVE) y cargar de nuevo (LOAD) los dibujos intermedios utilizando la opción SCREEN$, y las matrices y vectores con la opción DATA. A continuación puede cargar las siguientes subrutinas hasta finalizar el diagrama.

4) Por desgracia, algunos programas no se ajustan a una memoria de 8,8K incluso realizando todas estas maniobras; este es el caso del algoritmo general de líneas ocultas para objetos no triviales, el programa de "cine" y el juego DEFENSA DE LA ISLA. Si desea realmente profundizar en el estudio de gráficos por ordenador con el Spectrum, le aconsejamos calurosamente que se plantee la adquisición de una memoria de expansión de 32K para su ordenador.

Apéndice B
Listados de programas Basic incluidos en la cinta

Damos a continuación una lista de los listados de programas BASIC que se pueden encontrar en la cinta que se acompaña. Estas subrutinas están dispuestas para funcionar en la versión 48K de Spectrum. La mayor parte de las mismas no requiere cambios para funcionar en 16K; sin embargo, si usted posee este tipo de máquina, deberá comprobar las alteraciones que se indican en los comentarios REM de los listados, así como leer el Apéndice A. Las únicas subrutinas escritas específicamente para 48K son: "cine", las cinco pantallas "esfera1" a "esfera5" y el juego DEFENSA DE LA ISLA.

Tal como se indica en el Apéndice anterior, observará que en el capítulo 6 deberá cargar (LOAD) "rutdiag" y encadenar (MERGE) tan sólo uno de los siguientes programas: "6.8", "6.9", "6.10&11" o "6.12". Si necesita más de un tipo diferente de gráficos de datos en pantalla en un momento determinado, deberá guardar (SAVE) los dibujos intermedios en cinta y volver a cargar (LOAD) la pantalla una vez creado el nuevo programa.

CARA 1

Nombre del fichero	*Contenido*
direct1	Directorio 1.
rut1	Listados 2.1, 2.2, 2.3, 2.4, 2.8 y 3.3. Subrutinas para realización de gráficos en el espacio bidimensional.

2.9	Listado 2.9; uniendo puntos de un N-ágono regular.
2.12	Ejemplo de un recubrimiento.
2.13	Espirógrafo.
3.1	Cuadrados en cuadrados en cuadrados en..., etc.
rut2	Listados 3.4, 4.1a, 4.2a, 4.3a, 4.4a, 4.5, 4.6. Subrutinas para la manipulación de matrices en dos dimensiones.
4.10	Subrutina de construcción general de una elipse.
4.11 } 4.12	Construcción de cuatro "naves espaciales".
7.1	Punto de intersección de una recta y un plano en el espacio tridimensional.
7.2	Punto de intersección de dos rectas en el espacio tridimensional.
7.3 & 4	Ejemplo de producto escalar y vectorial.
7.5	Inversa de una matriz 3 × 3.
7.6	Punto de intersección de tres planos en el espacio tridimensional.
7.7	Recta de intersección de dos planos en el espacio tridimensional.
rut3	Listados 3.4, 9.1, 9.2 y la versión eficiente de los listados 8.1, 8.2, 8.3 y 8.4; subrutinas para manipulación de matrices en el espacio tridimensional.
8.5 & 6	Rotación de un punto con respecto a un eje arbitrario.
9.6 & 7 & 8	escena3, subrutinas de construcción y dibujo; necesarias para la realización en proyección ortogonal de dos cubos.
9.9	Subrutinas adicionales para dibujar el reactor (ortogonal).
9.10 & 11	Subrutinas para construir una proyección ortogonal de un cuerpo de revolución.
10.1	Orientación de un triángulo en tres dimensiones.
9.3 & 10.2	escena3 y subrutina de superficies ocultas para dibujar un cubo.
10.6	Subrutina de superficies ocultas para dibujar un cuerpo convexo de revolución.
0.1	Subrutina de construcción de un platillo volante (!).
10.4	Subrutina de superficies ocultas para dibujo de una superficie matemática.
11.1 & 2	Proyección en perspectiva de dos cubos.
12.1	Algoritmo general de líneas ocultas.
12.2	Dos cubos en perspectiva con eliminación de líneas ocultas.
12.3 & 4	Subrutinas de construcción de cubooctaedro e icosaedro.
12.5 & 6 & 7	Subrutinas de construcción de dos estrellas, y una subrutina escena3.
cine	Programa para presentar cinco imágenes en rápida sucesión.

esfera1 esfera2 esfera3 esfera4 esfera5	Cinco encuadres requeridos por el programa anterior para producir una esfera en movimiento.

CARA 2

Nombre del fichero	*Contenido*
direct2	Directorio 2.
borrado renum	Utilidades del listado 13.9.
5.5	GENERADOR DE CARACTERES.
5.4	Programa principal para realización de teselados sencillos.
rutdiag	Listados 6.1, 6.4, 6.6; subrutinas básicas para la construcción de diagramas.
6.2	"papel" y "tinta".
6.3	"punto" y "línea".
6.5 & 7	"etiqueta", "caracteres reducidos" y "crear".
6.8	Subrutina de histograma/tipo 1.
6.9	Subrutina de histograma/tipo 2.
6.10 & 11	Subrutina de círculos porcentuales y sombreado.
6.12	Gráficas científicas.
car3	Caracteres almacenados en el juego número 3 utilizados para realización de símbolos de menor tamaño.
car4	Caracteres del juego número 4.
1.16	JUEGO DEL GUSANO.
5.6	Juego del MASTER MIND.
máster	Caracteres del programa anterior.
5.7	Tablero de Ajedrez.
ajedrez	Juego de caracteres del programa anterior.
14.1	Juego de la DEFENSA DE LA ISLA.
video	Juego de caracteres del programa anterior.
escena	Escenario del juego anterior.
1.4	Programa de muestra de curvas fractales.
1.5	Dibujos con OVER.
5.1	Caracteres en binario.
5.2	Programa de formación de *pixels* gigantes.
5.3	Construcción de caracteres gráficos a partir de ocho números binarios.
13.1	Entrada de datos para 10 vértices de un polígono.

13.6	"Scroll" de la pantalla: en código máquina.
13.7	Versión BASIC del 13.6.
13.8	Programa autolistante
14.4	Técnica de la cascada.

Indice alfabético

Indice de subprogramas incluidos en el texto

ANAYA MULTIMEDIA

Colección «MICROINFORMATICA»

PROGRAMACION AVANZADA EN BASIC.—Bishop, P.
PASCAL A PARTIR DEL BASIC.—Brown, P.
EL ORDENADOR PERSONAL: COMO ELEGIRLO Y UTILIZARLO.—Cavalcoli, A.
PROGRAMACION EN BASIC: UN METODO PRACTICO.—Dachslager, H.; Hayashi, M.; Zucker, R.
TU PRIMER LIBRO DEL ZX SPECTRUM.—Dewhirst, J.; Tennison, R.
ASTRONOMIA: EL UNIVERSO EN TU ORDENADOR (ZX Spectrum).—Gavin, M.
EL ORDENADOR Y TUS HIJOS.—Hammond, R.
EL LIBRO GIGANTE DE LOS JUEGOS PARA ORDENADOR.—Hartnell, T.
BITS Y BYTES: INICIACION A LA INFORMATICA.—Heller, R. S.; Martin, C. D.
MICROINFORMATICA. Conceptos básicos.—Hollerbach, L.
DESCUBRE LAS MATEMATICAS CON TU MICRO.—Johnson, D.
EL ORDENADOR EN EL AULA.—Pentiraro, E.
EL LIBRO DEL BASIC.—Zaks, R.

DISEÑO DE GRAFICOS Y VIDEOJUEGOS (ZX Spectrum).—Angell, I. O.; Jones, B. Y.
MATEMATICAS DIVERTIDAS EN BASIC.—Kosniowski, C.
INTELIGENCIA ARTIFICIAL. CONCEPTOS Y PROGRAMAS.—Hartnell, T.
¿QUE ES LA TELEMATICA? Nuevas tecnologías en la sociedad de la información.—Servello, F.
COMO SE PROGRAMAN LOS ORDENADORES. Programación estructurada básica.—De Rosso, V.
«SPRITES» Y GRAFICOS EN LENGUAJE MAQUINA (ZX Spectrum).—Durst, J.
EL LIBRO GIGANTE DE LOS JUEGOS PARA ZX SPECTRUM.—Hartnell, T.
LENGUAJE MAQUINA AVANZADO (ZX Spectrum).—Webb, D.
LOS ORDENADORES NO MUERDEN.—Coccione, L.; Winter, G.

NOTAS

El rápido avance de la tecnología ha permitido que actualmente existan ordenadores personales baratos con la potencia de cálculo y la definición gráfica necesaria para afrontar aplicaciones de Diseño Asistido por Ordenador o desarrollar videojuegos interactivos y dinámicos.

DISEÑO DE GRAFICOS Y VIDEOJUEGOS es un texto introductorio al campo de gráficos por ordenador, que es una de las áreas de más rápida expansión en el mundo de los ordenadores. Este libro constituye un paquete de «software gráfico» para tu Spectrum que te guiará hasta la geometría del espacio tridimensional, los algoritmos de líneas y superficies ocultas y la construcción y animación de complejos objetos tridimensionales. La última parte del libro ofrece ideas avanzadas de programación en BASIC y un ejemplo minuciosamente analizado de videojuego.

DISEÑO DE GRAFICOS Y VIDEOJUEGOS supone que posees un razonable conocimiento de BASIC y de la geometría cartesiana y que deseas aprender nuevas aventuras en el mundo de los gráficos por ordenador.



Si los gráficos por ordenador te atraen... ¡adelante!

